НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Програмна система для аналізу запитів природною мовою»

Виконав:
студент IV курсу, групи КП-51
Романюк Сергій Олександрович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н.,
Заболотня Т.М. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н.,
Дідковська М.В. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____Дичка І.А.

«___»_____ 2018 р.

ЗАВДАННЯ

на дипломний проект студенту

Романюку Сергію Олександровичу

1. Тема проекту «Програмна система для аналізу запитів природною мовою», керівник проекту Заболотня Тетяна Миколаївна, доцент, к.т.н., затверджені наказом по університету від «22» травня 2019 р. №1331-С.

2. Термін подання студентом проекту «__» червня 2019 р.

3. Вихідні дані для дипломного проектування: див. Технічне завдання.

4. Перелік задач, які потрібно вирішити:
   - підготувати базу даних питань і відповідей;
   - проаналізувати характеристики питань;
   - розглянути існуючі рішення для аналізу запитів природною мовою;
   - розробити структуру запитів та відповідей на них;
   - реалізувати алгоритм аналізу запиту природною мовою і пошуку відповіді;
   - реалізувати графічний інтерфейс для взаємодії користувача із програмною системою;
   - виконати тестування програмної системи.

5. Перелік обов'язкового ілюстративного матеріалу:
   - структурна схема програмної системи (креслення);
   - структура бази даних (креслення);
   - алгоритм побудови структурованого питання (плакат);
   - особливості архітектури модуля для роботи з базою даних питань та відповідей (плакат).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент, к.т.н.		

7. Дата видачі завдання «31» жовтня 2018 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	16.11.2018	
2.	Розробка та узгодження технічного завдання	09.12.2018	
3.	Підготовка матеріалів першого розділу дипломного проекту	30.12.2018	
4.	Розроблення структури запиту та відповіді	16.01.2019	
5.	Підготовка матеріалів другого розділу дипломного проекту	01.03.2019	
6.	Розроблення алгоритму аналізу запиту природною мовою і генерації відповіді	22.03.2019	
7.	Програмна реалізація та тестування програмної системи	12.04.2019	
8.	Підготовка матеріалів третього розділу дипломного проекту	25.04.2019	
9.	Підготовка матеріалів четвертого розділу дипломного проекту	02.05.2019	
10.	Підготовка графічної частини дипломного проекту	19.05.2019	
11.	Оформлення документації дипломного проекту	26.05.2019	

Студент _____ Романюк С.О.

Керівник проекту _____ Заболотня Т.М.

АНОТАЦІЯ

Даний дипломний проект присвячено створенню програмної системи для аналізу запиту, поставленого природною мовою, з використанням засобів обробки природної мови. Тема роботи обумовлена необхідністю зниження часу, який витрачає викладач на надання відповідей на часто задавані запити (питання), та автоматизації цього процесу.

Розроблена програмна система являє собою сервер для виконання аналізу вхідних запитів і роботи з підготовленою базою даних питань та відповідей, який також містить логіку взаємодії з прикладним програмним інтерфейсом платформи Telegram для використання інтерфейсу чату на даній платформі. Аналіз вхідних запитів відбувається за допомогою використання натренованої моделі для виділення ключових слів. Ця модель базується на алгоритмі TFxIDF. Функціональність системи забезпечує доступ зареєстрованих на платформі Telegram користувачів до чату з ботом, дозволяючи ставити йому запитання, та отримувати на них відповіді, знайдені у базі даних питань та відповідей або у інших опціональних джерелах інформації. Інші способи доступу до функціоналу системи, окрім як через платформу Telegram, відсутні, однак наявна архітектурна можливість їх додавання. Відповідь на поставлене запитання, в залежності від його складності, може бути знайдена або у вищезгаданій базі даних, або у інших джерелах даних за допомогою додаткових систем, якщо наявна така інтеграція.

У даному дипломному проекті розроблено: архітектуру системи, алгоритм виділення ключових слів із запитання, заданого природною мовою, процедуру підбору релевантної відповіді з наявних джерел даних та підготовлено базу даних часто задаваних питань та відповідей на них.

ABSTRACT

This diploma project is dedicated to the creation of a software system which analyzes the request, queried in natural language, using natural language processing tools. The topic of the work is caused by necessity of decreasing the time teacher has to dedicate to answer the frequently asked requests (questions) as well as automatizing the process.

The created software system is a server that performs the analysis of an input data and works with a predefined questions and answers database, as well as using the Telegram API for accessing the chat interface on it. The system's functionality allows users who are registered and authenticated in Telegram to use a chat with the bot, where they can ask and receive answers, found in questions and answers database or other optional information sources. There are no ways to access the system's functionality other than through Telegram, but the architecture built allows such extensions. The answer for the question depending on its complexity could be found in the aforementioned database or in other information resources via extra systems, if such are integrated into main system.

The following is developed in the terms of this project: the architecture of system, the keyword extraction algorithm (for the question asked in natural language), the procedure of choosing a relevant answer from the available information resources and also the database of frequently asked questions and answers to them.

ДП.045440-01-90 Програмна система для аналізу запитів природною мовою.
Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Програмна система для	5	
	аналізу запитів		
	природною мовою.		
	Технічне завдання		
ДП.045440-03-81	Програмна система для	57	
	аналізу запитів		
	природною мовою.		
	Пояснювальна записка		
ДП.045440-04-51	Програмна система для	4	
	аналізу запитів		
	природною мовою.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Програмна система для	13	
	аналізу запитів		
	природною мовою.		
	Керівництво користувача		
ДП.045440-06-99	Програмна система для	1	
	аналізу запитів		
	природною мовою.		
	Структурна схема		
	програмної системи.		
	UML діаграма		
	компонентів.		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Програмна система для	1	
	аналізу запитів		
	природною мовою.		
	Структура бази даних.		
	ER діаграма		
ДП.045440-08-98	Програмна система для	1	
	аналізу запитів		
	природною мовою.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ПРОГРАМНА СИСТЕМА ДЛЯ АНАЛІЗУ ЗАПИТІВ ПРИРОДНОЮ МОВОЮ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ С.О. Романюк

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмна система для аналізу запитів природною мовою.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання в якості допоміжної інформаційної системи для викладача з метою зменшення навантаження на нього за допомогою надання студентам можливості відправляти запит та отримувати на нього релевантну відповідь, автоматично обрану з підготовленої бази даних часто задаваних питань та відповідей на них, або з іншого джерела даних.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Система повинна забезпечувати такі основні функції:

1) змога використання функціоналу системи через інтерфейс бота на платформі *Telegram*;
2) аналіз питання, заданого у текстовій формі природною мовою;
3) доступ до бази даних питань та відповідей;

4) перспектива розширення системи через додання джерел, де відбувається пошук інформації для відповіді на поставлені питання;
5) спроможність користувача оцінити релевантність наданої йому відповіді.

Розробку виконати на мові програмування *Python*.

Додаткові вимоги:

1) фокусування на тематиці питань з предмету “основи програмування”;
2) локалізація російською мовою;
3) додання двох або більше додаткових джерел для пошуку інформації на питання;
4) можливість, за потреби, перенаправити питання реальній людині (або групі людей) та зберегти до БД надану відповідь, якщо вона виявилася корисною;
5) розширення тематики питань, які можуть бути поставлені системі;
6) додання додаткових паралельно працюючих локалізацій англійською та українською мовами;
7) інтегрування функціоналу системи у існуючий веб-сайт та/або надання доступу до інтерфейсу системи через власне розроблений веб-сайт.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;

3) керівництво користувача;

4) креслення:

– «Структурна схема програмної системи. UML діаграма компонентів»;

– «Структура бази даних питань та відповідей. ER діаграма».

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи……..16.11.2018

Розробка та узгодження технічного завдання……..09.12.2018

Підготовка матеріалів першого розділу дипломного проекту.....30.12.2018

Розроблення структури запиту і відповіді……16.01.2019

Підготовка матеріалів другого розділу дипломного проекту……...……..01.03.2019

Розроблення алгоритму аналізу запиту природною мовою і генерації відповіді ……22.03.2019

Програмна реалізація та тестування програмної системи……..……..12.04.2019

Підготовка матеріалів третього розділу дипломного проекту ……25.04.2019

Підготовка матеріалів четвертого розділу дипломного проекту …….……...02.05.2019

Підготовка графічної частини дипломного проекту ……...20.05.2019

Оформлення документації дипломного проекту……..26.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ПРОГРАМНА СИСТЕМА ДЛЯ АНАЛІЗУ ЗАПИТІВ ПРИРОДНОЮ МОВОЮ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ С.О. Романюк

2019

ЗМІСТ

ВСТУП

У процесі розвитку людини, більша частина знань отримується нею за допомогою обміну інформацією з іншими людьми. Це відбувається напряму – особисті зустрічі та спілкування, або дистанційно – за допомогою електронного листування. Інший спосіб почерпнути знання іншої людини – ознайомитися з її роботою – книгою, дисертацією або статтею. Даний спосіб передбачає те, що автор витратить певну кількість часу на створення рукопису, але, разом з цим, зможе поділитися своїми знаннями з багатьма. Однак, не дивлячись на те, що даний спосіб є досить прийнятним, якщо потрібно поділитися певними знаннями з великою кількістю людей, індивідуальні або групові підходи є найбільш плідними для навчання людей.

Так як не завжди є достатньо часу на те, щоб поділитися усіма необхідними знаннями індивідуально з кожним бажаючим, з вищеописаних способів створюють оптимальний симбіоз. Так, наприклад, у магазині майже завжди присутні описи товарів, і кожна людина може ознайомитися з потрібною для неї інформацією. Але, водночас, часто є консультанти, особливо в технічних або спеціалізованих магазинах. Вони мають відповідний рівень знань у області товарів, які пропонуються на продаж, і готові допомогти покупцям, щоб останні змогли обрати найбільш придатні для них товари.

Також, часто у різних людей виникають однакові або доволі схожі питання. Щоб спростити процес отримання необхідних для конкретної людини знань, створюють списки часто задаваних питань – та надають на них підготовлені докладні відповіді, розміщуючи їх на сайтах компаній або магазинів. Окрім допомоги кінцевому користувачу, це також знижає неефективну частку роботи для консультантів, яким, інакше, доводиться надавати одні й ті ж відповіді різним людям у різний час.

Іншим прикладом є процес навчання у закладах освіти, особливо у вищих навчальних закладах. У відношеннях між викладачем і студентами постає проблема часу, який може бути приділений кожному індивідуальному студенту. Тут також застосовується вищезгаданий симбіоз: де це можливо, викладачі індивідуально працюють зі студентами, кожний з яких потребує різної кількості часу для засвоєння певної теми. Але пройти увесь необхідний матеріал з кожним індивідуально не є фізично можливим. Окрім того, що зі студентом треба пройти матеріал, указаний у методичній книжці, його також слід обробити таким чином, щоб індивід зрозумів його, і, окрім цього, надати поради відносно того, до яких джерел інформації він може звернутися, щоб згадати пройдений матеріал або глибше дослідити його. Щоб упевнитися, що усі студенти засвоїли необхідний матеріал, окремі теми виносяться на повторення на заняттях, або проводяться додаткові консультації.

Часто буває, що у вільний від занять час різні студенти звертаються до викладача у різний час, задаючи йому одні й ті ж питання – іноді особисто, іноді – через електрону пошту або месенджери. І викладачу, у кращому випадку, доводиться, один раз відповівши на питання, копіювати відповідь на нього або пояснювати його тими самими словами усім студентам, або, у гіршому випадку, для кожного студента пояснювати його наново, підбираючи інші слова. Хоча, з іншої сторони, можна, маючи певний історичний досвід того, які питання задаються студентами із року в рік, підготувати базу даних відповідей на ці питання, і надати студентам інтерфейс доступу до неї – тим самим, автоматизувавши цей процес, як у розглянутому випадку з консультантами.

Отже, побудова системи, яка б автоматично надавала відповіді на задавані студентами питання з певної дисципліни – це актуальна задача.

Даний дипломний проект і присвячено розробленню системи, яка призначена для допомоги викладачу у його діяльності, надаючи відповіді

на питання, які часто задають студенти, зменшуючи навантаження викладачів.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

БД *(DB, Database)* – база даних;

ОС – операційна система;

КПІ ім. Ігоря Сікорського – Київський Політехнічний Інститут імені Ігоря Сікорського;

ПЗ – програмне забезпечення, програмний застосунок;

СКБД *(DBMS, Database Management System)* – система керування базами даних;

Q&A System (Questions and Answers System) – інформаційна система, яка виконує обробку питань природною мовою. Аналізуючи запити, система надає на них пряму відповідь, знайдену у певному сховищі даних, або генерує її, використовуючи інформацію з декількох джерелах інформації;

API (*Application Programming Interface*, прикладний програмний інтерфейс) – набір функцій, методів, протоколів взаємодії з певним програмним застосунком;

AI (Artificial Intelligence) – штучний інтелект;

ML (Machine Learning) – машинне навчання;

DL (Deep Learning) – глибоке навчання;

NLP (Natural Language Processing) – оброблення природної мови (тексту);

Back End – це серверна частина, яка виконує бізнес логіку та є пластом доступу до даних;

Front End – це клієнтська частина, абстракція над пластом *Back End*, яка надає доступ до розташованого за ним функціоналу через інтерфейс, зручний для користувача;

NN (Neural Network) – нейронна мережа;

ANN (Artificial Neural Network) – штучна нейронна мережа;

MLP (Multilayer perceptron) – багатошаровий перцептрон;

CNN (Convolutional Neural Network) – згорткова нейронна мережа;

RNN (Recursive Neural Network, Recurrent Neural Network) – рекурсивна нейронна мережа або рекурентна нейронна мережа;

LSTM (Long short-term memory) – довга короткочасна пам’ять;

SNN (Shallow Neural Network) – неглибока нейронна мережа;

POS tagging (Part-of-speech tagging) – це розмітка текстових даних; пов’язування слів з текстового набору з відповідними ним частинами мови;

NER (Named Entity Recognition) – розпізнавання у наборі текстових даних імен або унікальних назв, сутностей;

TF-IDF (Term Frequency – Inverse Document Frequency) – статистична величина, яка показує оцінку важливості слова у контексті певного документу. Є добутком величин *Term Frequency* та *Inverse Document Frequency*;

TF (Term Frequency) – частота слова – відношення кількості використань певного слова до загальної кількості слів у документі, який обробляється;

IDF (Inverse Document Frequency) – обернена частота документа – це інверсія частоти, з якою певне слово зустрічається у всіх документах певної колекції.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз процесу обробки запитів, заданих природною мовою

Під час розробки даного дипломного проекту проаналізовано процес обробки запитів, які формують люди звичним для них чином – природною мовою.

Загалом, поняття запиту має декілька визначень. Запитом можна назвати команду визначеного формату, яка передається системі у вигляді певного ключового слова. Програма, обробляючи команду, видає певний очікуваний від неї результат.

Іншим прикладом є написання та виконання програмного коду. Код – це, по суті, набір запитів до певної програми. За умови коректного формування даного набору та передання його на обробку відповідній системі, користувач – програміст – отримує шуканий результат. Результатом може слугувати отримана після проведення ряду математичних розрахунків відповідь. Це також можуть бути метрики, отримані після оброблення певного набору даних, або інформація, знайдена системою у якомусь джерелі даних.

Усі види запитів об’єднує наступна характеристика: вони робляться для того, щоб отримати певний результат.

Розглянемо останній наведений приклад більш детально. Коли людина прагне знайти певну інформацію, вона, зазвичай, формує питання – запит – і прагне знайти на нього відповідь. Коли питання ставиться іншій людині, вона його аналізує та формує відповідь, яка на її думку, зможе надати ту інформацію, яку шукає запитувач.

Автоматизація даного процесу означає вилучення людини, яка надає відповідь, і заміни її програмною системою, яка змогла б сама виконати описані вище дії.

Таким чином, процес відповіді можна в цілому розділити на такі етапи:

- отримання запиту від користувача та аналіз заданого питання;
- знаходження інформації, яка так чи інакше може бути релевантною в розрізі цього питання;
- підготовка відповіді та надання її користувачу.

Перший етап включає в себе використання технологій оброблення природної мови. Результатом аналізу тексту питання є виділення теми або підтеми, ключових слів, за якими можна буде здійснити пошук необхідної інформації рядом способів. До них належать пошук у підготовленій базі даних, семантичний пошук у корпусі текстових даних, використання *API* спеціалізованих сервісів та інші.

Варто зауважити, що питання може задаватися у різний спосіб. Це може бути як “класичне” у розрізі людської комунікації питання, сформоване звичним чином:

- *“Відповіси, будь ласка, які властивості у цілочисельного типу даних?”;*

Так і спеціально сформований самим користувачем запит, який є набором ключових слів (що є більш простим для програми):

- *“цілочисельний тип даних опис”;*

Або навіть складене з декількох речень питання, яке може бути розділене на окремі запити:

- *“Що таке цілочисельний тип даних? Для чого його застосовують? Які його властивості?”.*

Другий етап застосовує науки інформаційного пошуку та інтелектуального аналізу даних. Підходи або технології, які стосуються цих сфер, застосовуються для аналізу неструктурованих даних з метою пошуку інформації в них, виявлення залежностей між ділянками текстових даних, виділення релевантних даних та інших.

Пошук інформації може відбуватися у низці джерел. Це може бути як наперед сформована база даних конкретних питань та відповідей, так і об’ємна система, яка вміщає у собі великі за обсягом неструктуровані або частково структуровані текстові дані по певній тематиці; набори довільних даних (наприклад, діалоги між людьми на сайтах, системах питань та відповідей) та семантичний пошук по мережі у цілому.

Останнім етапом є аналіз усього набору інформації, який сформовано на попередньому етапі, та виділення з нього найбільш релевантних відповідей, засновуючись на низці параметрів, таких як відповідність тексту питання та відповіді за виділеними словами, популярності надання того чи іншого набору інформації (наприклад, якщо надана у діалозі відповідь оцінювалася позитивно великою кількістю людей, або стаття, з якої виділено відповідну інформацію, має велику кількість переглядів) тощо, підготовка відповіді належним чином і відправка її запитувачу.

1.2. Огляд існуючого програмного забезпечення для аналізу запитів природною мовою

В рамках виконання даного дипломного проекту також досліджено можливості існуючих програмних засобів, які можуть проводити аналіз запитів, заданих природною мовою. В основному, такі системи призначені для надання відповідей на задані ним запити-питання. Такі системи є тісно пов’язаними з аналізом тексту та, іноді, виділенням інформації з неструктурованого текстового корпусу даних. Метою проведеного огляду було дослідити існуючі програмні продукти, призначені для автоматичного надання відповідей, і з’ясувати, наскільки вони відповідають вказаним в задачі вимогам.

1.2.1. Slackbot (вбудований у Slack [1])

Даний приклад є схожим до системи, побудованої у рамках даного проекту, з точки зору взаємодії з користувачем. *Slackbot* є одним з супутніх компонентів системи *Slack*. Остання, в свою чергу, є платформою, яка використовується для комунікації між людьми однієї команди, які працюють над певним проектом. Більша частина спілкування відбувається через інтерфейси чатів.

Slackbot – це бот, який слугує для того, щоб надавати допоміжну інформацію користувачам про платформу *Slack*, її інтерфейс та функції. Взаємодія з ботом відбувається у вигляді відправлення йому повідомлень через інтерфейс чату – у такий же спосіб, який надається для обміну повідомленнями між користувачами. Боту можна задавати як граматично коректні питання, наприклад: "*Як відправити повідомлення?*", так і просто вказувати у запиті тільки ключові слова: "*Повідомлення відправка*". Після одної або двох (в окремих випадках – до п'яти) секунд очікування бот надає відформатовану відповідь, яка, згідно його аналізу, надає інформацію, пов'язану з питанням. Якщо ж бот не зміг знайти відповіді на це питання, він сигналізує про це відправкою відповідного повідомлення, яке включає в себе посилання на головну сторінку довідкового центру. Приклад взаємодії з цим ботом наведено нижче, на рис. 1.

Як можна побачити з діалогу на рисунку, бот досить погано справляється з іншими мовами, окрім англійської. Вона є основною, але також (судячи за відповідями) частково підтримується і російська. Але це, скоріш за все, не стільки підтримка мови, скільки підтримка кодування і розпізнавання слів, які пишуться в обох мовах по-однаковому (наприклад, "status" та "статус".

Інколи бот дає вичерпні відповіді, які не потребують додаткових посилань. Деякі ж відповіді, навпаки, не мають у своєму тілі корисної інформації, що відноситься до питання, і тільки надають короткий опис наданого посилання на конкретну інформацію у довідковому центрі.

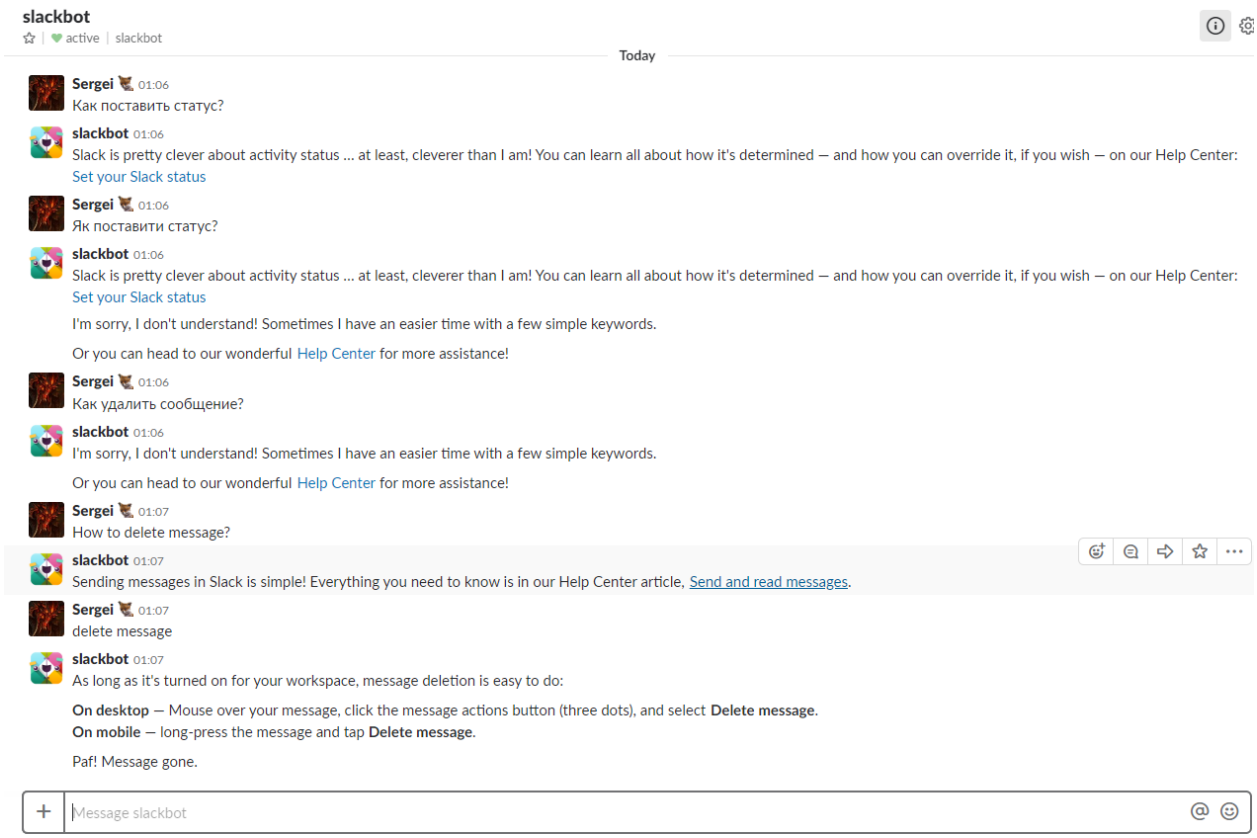


Рис. 1. Взаємодія зі *Slackbot*.

При цьому, іноді бот видає різні відповіді на майже однакові запити. Так, на запит “*How to delete message?*” (повне питання) бот відповідає інструкцією про те, як *відправити* повідомлення, а на запит “*delete message*” (використання ключових слів) бот дає коректну підказку про те, які дії потрібно виконати для видалення повідомлення.

Таким чином, можна виділити такі сильні та слабкі сторони бота:

+ *легкий у використанні*: взаємодія з ботом відбувається через звичний користувачам сервісу чат;
+ *додаткові можливості*: доступний також і всі супутні можливості чату: пересилання відповідей бота, їх видалення, збереження до “Закладок” тощо;
+ *часткова підтримка декількох мов (підтримка символів)*: хоча і не можна сказати, що бот повноцінно працює з декількома мовами, але він підтримує символи кирилиці;

+ *допомога ззовні*: навіть якщо бот не може знайти відповідь самостійно, він пропонує скористатися довідковим центром у якості крайнього варіанту;
– *слабкий алгоритм аналізу питання*: як встановлено емпіричним шляхом, бот, навіть при мінімальних змінах текста запиту, які не змінюють суть питання, видає різні результати;
– *вузька сфера застосування*: бот може відповідати тільки на питання по *Slack*. Даний бот і створено з метою відповідати тільки на питання, які стосуються даної сфери. Однак в цілому для питально-відповідальної системи це мінус;
– *обмежена БД*: скоріш за все, у бота є сформована наперед база даних питань та відповідей. Якщо він не знаходить відповіді на задане питання, він не намагається знайти відповідь у сторонніх ресурсах, а просто перенаправляє користувача до довідкового центру для самостійного пошуку.

1.2.2. Jill Watson [2] (Ashok K. Goel, Georgia Institute of Technology)

Даний приклад є схожим до системи, побудованої в рамках даного проекту, з точки зору мети створення. *Jill Watson* – ім'я бота, якого створив професор Технологічного інституту Джорджії *Ashok K. Goel*.

Jill створена для того, щоб знизити навантаження на викладачів, автоматично надаючи студентам відповіді на поширені питання. Таким чином, викладачі не витрачали свій час на відповіді на одні і ті самі питання, а могли зайнятися більш корисними для вчительської діяльності речами. Початковий прототип *Jill* створено на основі нейронної мережі, яку навчено "алгоритмом навчання з викладачем". Спочатку, була зібрана база даних питань та відповідей за декілька попередніх років та передана *Jill* для обробки. Потім ці питання були розсортовані за декількома категоріями, і *Jill* навчили правильно визначати категорію одержуваного

питання. Після цього, *Jill* навчили знаходити відповідь на питання серед інших питань класифікованої категорії, і, якщо відсоток впевненості у результаті роботи алгоритму сягав 97% або більше, приймати знайдену відповідь за правильну і надавати її користувачу.

Після розробки, даний прототип вирішено додати до основного форуму, де студенти ставили питання. *Jill* давала відповіді приблизно на 10% питань, але не всі відповіді були корисними, а деякі були загалом неправильними, адже AI неправильно класифікувала категорію питання. Після цього, було прийняте рішення створити дзеркальний форум, на який було перенесено *Jill*. Питання з основного форуму перенаправлялися також і до віддзеркаленого форуму, там бот надавала відповідь, і потім, людина, власноруч аналізуючи цю відповідь, перенаправляла її на основний форум, якщо вона була правильною.

За словами професора А. Гоела, продуктивність першої версії бота була досить низькою. Вони спробували перекласифікувати увесь набір даних, враховуючи нові надані ботом відповіді, перетренували її на новому наборі даних, і результати дещо покращилися, але цього було недостатньо. Після цього, командою розробки прийнято рішення створити свою власну модель, яка б, окрім аналізу раніше наданих відповідей, додатково включала би “контекст і структуру взаємодії *Jill* з зовнішнім світом, зі студентами” [3]. Після додання цієї моделі, результати *Jill* значно зросли, і, коли правильність надаваних відповідей сягнула відмітки в 97%, дзеркальний форум було прибрано, і *Jill* була переміщена на основний форум, де вона успішно справлялася зі своїм завданням. При цьому, *Jill* відповідала на питання, які вона класифікувала як питання з дисципліні “Комп’ютерні науки”, а на інші питання вона відповідала порожнім повідомленням.

На жаль, про майбутнє даного проекту додаткової інформації знайти не вдалося, окрім того, що розробники планують створити Jill наступної версії, яка відповідатиме на питання значно більшого спектру.

Отже, сильні і слабкі сторони даного проекту:

+ *успішний алгоритм підбору правильної відповіді*: другий створений прототип показав високий відсоток правильних відповідей у 97%;
+ *якісний алгоритм аналізу питання*: згідно з результатами, якими ділився професор А. Гоел після закінчення тестової стадії проекту [4], алгоритм однаково класифікував питання, задані по-різному, але які несли у собі однаковий сенс;
+ *не потребує додаткових дій від користувача*: *Jill* оперувала одночасно зі всією навчальною платформою (усім веб-форумом), і коли надходило питання – давала на нього відповідь;
+ *відсутність перешкоджання процесу навчання*: коли Jill була додана до форуму, ніхто не здогадувався, що *Jill* – це не реальна людина, а AI, поки професор А. Гоел сам не поділився результатами проекту зі студентами;
– *обмежена тематика*: хоча згодом *Jill* надавала досить велику кількість правильних відповідей, майже замінив живих викладачів у цій задачі, вона все одно давала відповіді тільки на питання, які стосувалися теми дисципліни, що вивчалася;
– *локалізація*: взаємодія з *Jill* відбувалася тільки англійською мовою;
– *великий інформаційний ресурс і затрати часу*: ціною того, щоб отримати успішно натреновану модель, є те, що у свій час вона тренувалася на великому корпусі даних, який треба збирати достатньо велику кількість часу, і потім очікувати, поки сама модель зможе обробити таку кількість інформації і натренуватися на ній.

1.3. Результати проведеного аналізу

Порівняно два доволі близьких до розробленого у рамках даного проекту програмні продукти, які аналізують запити, задані природною мовою:

- *Slackbot*;
- *Jill Watson*.

Отже, ознайомившись з перевагами і недоліками обох проектів, можна зробити висновок, що *Jill* показала себе краще.

Переважна кількість наданих *Jill* відповідей, не дивлячись на те, що вони були згенерованими, а не підготовлені завчасно – була правильною і корисною для користувачів. Відповіді, які вона продукувала, майже завжди задовольняли запитувачів повною мірою: вони дякували *Jill* за відповідь і не задавали інших уточнюючих питань. Відповідно, реальній людині не потрібно втручатися у процес взаємодії, – що і було бажаним результатом створення системи.

Якість наданої інформації є основним критерієм оцінки системи, яка аналізує питання і знаходить до нього релевантну відповідь. *Slackbot,* як видно на прикладі взаємодії з ним, може впоратися з такою задачею часто тільки при вживанні додаткових дій з боку користувача.

При цьому, аналізуючи недоліки обох систем, можна виділити основні обмежувальні критерії подібних проектів (як мінімум, на їх перших версіях):

- локалізація – одна мова;
- вузька спеціалізація / сфера застосування. Чим ширша сфера застосування – тим менша успішність алгоритму надання відповідей і тим складніше таку систему навчити: для цього потрібно значно більше даних та часу.

Отже, в результаті проведеного порівняння даних програмних продуктів, визначено, що кожен з подібних програмних продуктів

створюється зі своєю унікальною метою, і наразі не існує універсального рішення для усіх систем такого роду. Таким чином, можна дійти висновку, що жоден з існуючих програмних продуктів не задовольняє вимогам, що передбачені технічним завданням цього дипломного проекту.

2. АНАЛІЗ ТЕХНОЛОГІЙ СТВОРЕННЯ СИСТЕМ ДЛЯ АНАЛІЗУ ЗАПИТІВ ПРИРОДНОЮ МОВОЮ

2.1. Обгрунтування вибору месенджера *Telegram* у якості платформи для взаємодії з користувачем

Під час створення будь-якого програмного застосунку, перед розробником постає ряд задач. Одна з найважливіших задач – це формування інтерфейсу системи. Інтерфейс – це набір графічних елементів, через які, у рамках даної роботи, користувач зможе формувати запити і отримувати результати обробки цих запитів.

Архітектурно систему можна зробити монолітною або розподіленою. Розподілена система є більш прийнятною у рамках даної роботи, адже вона дозволить централізовано зберігати і оновлювати базу даних. Найпростішим випадком роздоліленої системи є така, що складається з двох частин: Back End та Front End.

У якості Back End виступає сервер з кодовою базою для аналізу питання й формування відповіді. У рамках цього серверу було реалізовано невелике API для доступу до зазначених функцій. Front End складова комуніціює з Back End на обох етапах передачі інформації (прийняття запиту від користувача та доставка сформованої відповіді) та відображає результати роботи.

Front End складову – інтерфейс – можна реалізувати у рамках десктопного додатку, мобільного застосунку або окремого веб-ресурсу. Останній краще за перші два, так як він не потребує встановлення ніяких додатків окрім браузера, і, отже, не є прив’язаним до конкретної платформи. З іншої сторони, з точки зору інтерфейсу, у межах веб-застосунку він буде однаковим, з, можливо, невеликими відмінностями в залежності від використовуваного браузера. Інтерфейс застосунків, які створюються індивідуально під кожну ОС, заздалегідь передбачається як більш зручний та звичний, адже він може бути написаний з використанням

нативних для платформи розробки функцій та бібліотек. Але такий підхід означає більші витрати, або фінансові, у вигляді набору різних груп розробників для різних ОС, або часові, якщо єдина команда має розробити декілька застосунків. Окрім цього, після закінчення розробки, кожен зі створених програмних засобів треба підтримувати. Слід як аналізувати звіти про помилки від користувачів та вносити зміни з приводу виправлення старих функцій, так і розробляти нові – окремо для кожної ОС. Веб-застосунок також потребує підтримки після створення, але вона, скоріш за все, буде простішою, в тому числі з точки зору часових витрат, адже організовувати підтримку для різних типів браузерів легше, ніж для різних ОС.

Слід зауважити, що вже існують програмні платформи, які дозволяють створювати інтерфейс для подібної системи у вигляді чат-ботів – месенджери. Вони, зазвичай, мають основу для свого графічного інтерфейсу як у вигляді веб-версії, так і у вигляді нативних десктопних та мобільних застосунків.

Таким чином, було прийняте рішення реалізувати Front End у вигляді чат-бота. Месенджером, у рамках якого розроблявся бот, обраний *Telegram*. Окрім цього сервісу, також розглядалися *Facebook Messenger*, *WhatsApp* або *Viber*.

Два основних критерії, які сформували остаточний вибір месенджера, – це популярність серед користувачів та доступність API для розробки чат-ботів на платформі сервісу.

Платформа *Viber* була виключена майже одразу, адже, не дивлячись на те, що месенджер є популярним і API для розробки ботів якісно задокументоване, з 1 квітня 2019 року компанія прийняла рішення зробити чат-ботів платними. Ціна за можливість відправлення повідомлень чат-ботами стартує від 4500$ за місяць [5].

Порівняння API трьох інших месенджерів наведено за посиланням [6]. *Twilio API*, яким представлене *WhatsApp*, не є популярним

та у наведеному порівнянні не представлено ніяких плюсів даного API. Для *Facebook Messenger* розробники виділяють можливість легкої інтеграції між API та їх сервісом. Для *Telegram* наведена низка плюсів, серед яких легка інтеграція з іншими сервісами, відмінна та легко зрозуміла документація, швидке налаштування та ін.

Зокрема, у статті за посиланням [7] наведене порівняння чат-ботів на платформах *Telegram* та *Facebook Messenger*, у якій сказано, що більшість ботів у останньому месенджері – незручні у використанні, що каже про те, що API може бути дещо складним для створення інтуїтивно зрозумілої взаємодії з чат-ботом.

Telegram є популярним серед молоді та студентів. Результатом проведеного опитування серед студентів КПІ ім. Ігоря Сікорського, зокрема факультету прикладної математики, є наступна статистика:

Рис. 2. Статистика відносно основного месенджера

Як можна побачити із зібраної статистики, переважна кількість респондентів обрала саме *Telegram* у якості свого основного месенджера. 326 з 359 відповідей надавалася студентами КПІ ім. Ігоря Сікорського, з яких 114 студентів (35%) – з першого курсу, 66 (20%) – з другого курсу, та решта – зі старших курсів. З ФПМ надали свої голоси 35

студентів. Усі студенти даного факультету одноголосно обрали *Telegram* у якості свого основного месенджера.

2.2. Порівняння мов програмування *Java* та *Python*

2.2.1. Мова програмування Java

Java – об’єктно-орієнтована мова програмування, яка розроблюється компанією *Sun Microsystems* [8]. Сфера її застосування значно широка: на ній пишуться т. наз. аплети (*applet*), застосунки та серверне ПЗ.

Java-аплети – це невеликі застосунки, написані на мові програмування *Java* (або на інших мовах програмування), які в подальшому компілюються у байт-код *Java* і надається користувачам у такій формі.

Програми на Java, як і аплети, можуть бути трансльовані у байт-код, який виконується на віртуальній *Java*-машині (*Java Virtual Machine*) – програмі, яка обробляє байт-код та передає інструкції обладнанню, так як інтерпретатор. Однак її перевагою є те, що байт-код, на відміну від текста, обробляється значно швидше.

Одна з найважливіших технічних вимог *Java* – платформна незалежність, що дозволить створювати програми, які не потребують компілювання для кожної окремої архітектури, і їх можна буде використовувати на різних процесорах у різних ОС.

Більшість архітектурних рішень, які були прийняті під час створення *Java*, були продиктовані тим, що потрібно було надати синтаксис, схожий з *C* та *C++*, і, водночас, розширити можливості, надавані у даних мовах. У *Java* використовуються практично ідентичні погодження для оголошення змінних, передачі параметрів, операторів та для управління потоком виконання коду. *Java* перейняла кращі риси цих двох мов.

Для мови *Java* можна виділити такі три ключових елементи:

- *Java* надає свої аплети для широкого застосування. Так як вони є невеликими, надійними, динамічними та платформо незалежними, вони добре підходять для розробки активних мережевих додатків, які вбудовуються у Web-сторінки;
- *Java* вміщує у собі силу об'єктно-орієнтованої розробки додатків, поєднуючи простий та знайомий синтаксис з надійним та зручним у роботі середовищем розробки. Це дозволяє широкому колу програмістів швидко створювати нові програми та аплети;
- *Java* як інструмент надає програмісту великий набір класів об'єктів для чіткого абстрагування багатьох системних функцій, які використовуються при роботі з вікнами, мережею та засобами вводу та виводу. Ключова риса цих класів є те, що вони забезпечують створення незалежних від використовуваної платформи абстракцій для широкого спектру системних інтерфейсів.

2.2.2. Мова програмування Python

Python – це високорівнева інтерпретована інтерактивна та об'єктно-орієнтована скриптована мова програмування, розроблена Гвідо Ван Россумом (Guido van Rossum) [9]. Синтаксис *Python* є надзвичайно простим для створення та читання коду – іноді здається, що використовується не синтаксис мови програмування, а синтаксис англійської мови, адже у *Python*, на відміну від інших популярних мов програмування, використовуються англійські ключові слова замість пунктуаційних символів у якості операторів.

Python є інтерпретованою мовою програмування, що означає, що він обробляється прямо під час виконання. Програму, написану на цій мові, не потрібно компілювати перед тим, як запустити її.

Python – інтерактивна мова програмування, і це означає, що можно у реальному режимі взаємодіяти з командним рядком *Python* та, таким чином, створювати програму “на льоту”.

Об’єктна орієнтованість мови *Python* означає те, що вона підтримує об’єктно-орієнтований стиль програмування, та підхід у програмуванні, який інкапсулює код усередині об’єктів. Взагалі, у мові *Python* все є об’єктом, в тому числі й виключення (Exception), засоби для роботи з примітивними типами даних (float, integer, …), і навіть функції. Останні є “об’єктами першого класу” (first-class object), що означає те, що їх можна передавати у якості аргументів до інших функцій, зберігати в змінні або навіть зберігати у окремих структурах даних.

Python є хорошою платформою для проекту майже будь-якого направлення, починаючи з інтерактивних веб-сайтів та ігор і закінчуючи складними науковими завданнями, серед яких – оброблення довільної текстової інформації.

У даної мови програмування багато особливостей, серед яких можна виділити наступні:

- easy-to-read – код, написаний з використанням стандартних ключових слів на *Python*, дуже легко читається і легко зрозуміти логіку, яка закладається розробником у код;
- У *Python* наявний дуже великий набір стандартних функцій та бібліотек, причому переважна більшість з цього набору є портативною і майже платформо незалежною (з деякими мінорними відмінностями) – підходить для UNIX-платформ, Windows та Macintosh. Деякі труднощі (відмінності), як може зазначити автор дипломного проекту, виникають при роботі з інструментами мови для криптографії, мережею та файловими шляхами за допомогою стандартних функцій. Однак на рахунок останнього, у *Python* 3.5 додано бібліотека *pathlib* як заміна стандартним функціям для роботи з файлами, яка нівелює

кросплатформні проблеми вбудованих функцій, і використання цієї бібліотеки рекомендується розробниками мови;

- Код, написаний мовою *Python* може використовуватися на великій кількості різноманітного комп'ютерного обладнання, і інтерфейс буде однаковим на кожній з платформ;
- У *Python* наявний інтерактивний режим, який дозволяє інтерактивно проводити тестування та відлагодження окремих ділянок коду;
- "Ядро" *Python*, за потреби, можна розширити за допомогою додання власноруч створених низькорівневих модулів, що дає більш тонкий контроль над ефективністю алгоритмів, які створює розробник;
- У *Python* реалізовані драйвера для усіх популярних БД, включаючи як SQL-орієнтовані *MySQL, PostgreSQL, MSSQL, SQLite*, так і NoSQL, серед яких *MongoDB, DynamoDB, Redis, Neo4j* та інші;
- *Python* є досить гарно масштабованою, що означає, що ця мова надає гарну структуру та підтримку для об'ємних програмних застосунків.

Серед більш технічних моментів можна виділити також такі:

- *Python* надає доступ до дуже високорівневих структур даних та підтримує динамічну перевірку типів даних. При коректному використанні цих можливостей, це дозволяє значно скоротити час, потрібний на реалізацію певних архітектурних рішень;
- *Python* можна використовувати як скриптовану мову програмування, так і компілювати її у байт-код для будування великих застосунків.
- У *Python* реалізований автоматичний "збирач сміття" (garbage collector). При цьому, за потреби, до його функцій можна

доступитися мануально за допомогою відповідних ключових слів мови;

- *Python* легко поєднати у розробці з іншими мовами програмування, такими як C/C++ (*Cython*), *Java* (*Jython*), *C#/.NET* (*IronPython*), *Common Lisp* (*CLPython*) та деякими іншими.

Базуючись на описах двох представлених вище мов програмування, обрано *Python* для розробки даного дипломного проекту. За допомогою доступного синтаксису та досить великого об'єму як базових, так і написаних спільнотою бібліотек, навіть складні технічні AI або NLP задачі можна вирішувати ефективніше та швидше, ніж у *Java*.

2.3. Опис підходів до аналізу запитів природною мовою

Аналіз запиту – це набір функцій, які слід застосувати до його тексту, щоб мати змогу сформувати коректний результат у відповідь на даний запит. Серед них – виділення ключових слів, перетворення слів на чисельні представлення (вектори) та ін. Це – задачі NLP, до вирішення яких здебільшого застосовуються ANN. Є велика кількість різновидів ANN, у середині яких – різні математичні підходи.

Далі буде описане базове представлення ANN та деякі з її різновидів, їх особливості та сфери NLP, у яких вони найбільше використовуються.

- *ANN*

Штучна нейронна мережа – це нелінійна обчислювальна модель, яка базується на нейронній структурі мозку, яку можна навчити виконанню завдань класифікації, прогнозування, прийняття рішень, візуалізації та ін.

ANN складається з штучних нейронів, або елементів обробки, і складається з трьох взаємопов'язаних пластів (шарів): вхідного, прихованого та вихідного.

Вхідний пласт складається з вхідних нейронів, які передають інформацію до прихованого пласту. Прихований пласт, у свою чергу, передає інформацію до вихідного. Кожен з нейронів має зважені входи

(синапси), функцію активації (функція, яка визначає вихідний сигнал, базуючись на вхідних даних – сигналах), та один вихід. Синапси є параметрами, які можна налаштовувати, які конвертують нейронну мережу в параметризовану систему.

Зважена сума входів виробляє сигнал активації, який передається до функції активації, щоб отримати вихідний сигнал з нейрона. Зазвичай, функції активації – лінійні, наприклад, кусково-неперервна функція кроку (функція "сходинки"), а також функції сигмоїди, гіперболічного тангенсу та ReLU ("випрамляч").

Тренування нейронної мережі (моделі) – це процес оптимізації ваг, який має на меті мінімізування проценту хибних прогнозувань, що приводить до визначеного проценту точності нейронної мережі. Метод, який найбільше використовується для визначення того, наскільки впливовим є кожен з нейронів на процент похибок називається методом зворотного поширення помилки *("backpropagation")*, і він підраховує градієнт функції втрат.

Систему можна зробити більш гнучкою та потужною за рахунок додання додаткових прихованих пластів. ANN з декількома прихованими пластами називається DNN, та за допомогою неї можна моделювати складні нелінійні взаємодії.

- *MLP*

Багатошаровий перцептрон – це ще один різновид ANN. Він має три або більше пластів. Він використовує нелінійну функцію активації (переважно гіперболічний тангенс або логістичну функцію), що дозволяє класифікувати дані, які не є лінійно відокремлюємими. Кожен вузол пласту пов'язаний з кожним вузлом з наступного пласту, і, таким чином, мережа є всебічно зв'язаною.

Прикладами задач NLP, які вирішуються даним різновидом ANN, є розпізнавання голосового вводу та машинний переклад.

- *CNN*

Згорткова нейронна мережа містить один або більше загорткових пластів, повністю пов’язаних між собою, і використовує варіацію MLP. Нейрони у згорткові пластах використовують функцію згортки до вхідних сигналів, передаючі результат функції до наступного пласту. Ця операція дозволяє зробити мережу більш глибокою зі значно меншою кількістю параметрів.

CNN показують видатні результати у задачах роботи зі зображеннями та мовленням. *Yoon Kim* у роботі *“Convolutional Neural Networks for Sentence Classification”* [10] описує процес та результати задач класифікації тексту, вирішених за допомогою CNN. Він описує модель, побудовану над *word2vec,* проводить над нею серію експериментів та оцінює її за допомогою певних бенчмарків, доказуючи, що модель показує відмінні результати.

У роботі *“Text Understanding from Scratch”* [11], *Xiang Zhang* та *Yann LeCun* демонструють, що CNN можуть досягнути видатної продуктивності навіть без розуміння слів, фраз, речень та будь-яких інших синтаксичних структур людської мови. Семантичний аналіз, виявлення парафраз та розпізнавання мови – задачі, які вирішуються за допомогою CNN.

- *RNN*

Рекурентна нейронна мережа, на відміну від класичної нейронної мережі, це варіація рекурсивної нейронної мережі, у якій зв’язки між нейронами утворюють спрямований цикл. Це означає, що результат функції – вихід – залежить не тільки від поточного входу, а й від результату аналізу попереднього входу. Ця пам’ять дозволяє користувачам такого підходу вирішувати такі NLP задачі як розпізнавання рукописного тексту або розпізнавання голосового вводу. У роботі *“Natural Language Generation, Paraphrasing and Summarization of User Reviews with Recurrent*

Neural Networks" [12], автори демонструють RNN модель, яка може генерувати речення у стилі новели або створювати підсумки документів.

Siwei Lai, Liheng Xu, Kang Liu та Jun *Zhao* створили RNN для класифікації тексту без описаних людиною ознак та описали це у роботі *"Recurrent Convolutional Neural Networks for Text Classification"* [13]. Їх модель порівняно з існуючими методами для класифікації тексту, та показано, що вона показує результати кращі за результати, які отримані традиційними методи, для будь-якого набору даних.

- *LSTM*

Довга короткочасна пам'ять – одна з різновидів архітектур RNN, яка спроектована для більш точного моделювання тимчасових послідовностей та їх далеких залежностей, ніж класичні RNN. LSTM не використовує функцію активації в межах своїх рекурентних компонентів, збережені значення не модифікуються, і градієнт не має тенденцію до зникнення під час тренування. Зазвичай, частини, з яких складається LSTM, об'єднуються в "блоках". Ці блоки мають три або чотири "воріт" (наприклад, вхідні ворота, ворота забуття та вихідні ворота), які керують потоком даних на основі логістичної функції.

В *"Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling"* [14], *Hasim Sak, Andrew Senior* та *Françoise Beaufays* показали, що глибокі LSTM RNN архітектури досягають найкращих сучасних результатів у акустичних моделюваннях великих масштабів.

У роботі *"Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network"* [15] авторів *Peilu Wang, Yao Qian, Frank K. Soong, Lei He,* та *Hai Zhao,* представлено модель для POS-tagging. Модель досягла точності у 97,40 %.

- *seq2seq (sequence-to-sequence)*

Зазвичай, модель "послідовність-до-послідовності" складається з двох RNN: енкодера, який обробляє вхідні дані, та декодера, який виробляє

вихідний результат. Енкодер та декодер можуть використовувати однакові або різні набори параметрів.

Seq2seq моделі здебільшого застосовують у Q&A системах, чат-ботах та машинному перекладі. Подібні багатошарові клітини успішно використовувалися у моделях для перекладу у роботі *"Sequence to Sequence Learning with Neural Networks study"* [16].

У *"Paraphrase Detection Using Recursive Autoencoder"* [17] представлена архітектура рекурсивного автоенкодера для новел. Результатом є вектори у n-мірному семантичному просторі, де фрази з подібним значенням є близькими одна до одної.

- *SNN*

Окрім DNN, є й неглибокі нейронні мережі, які також є популярним і використовуваним інструментом. Наприклад, *word2vec* – це група неглибоких двошарових моделей, які використовуються для створення векторного представлення слів. Представлена у *"Efficient Estimation of Word Representations in Vector Space"* [18], *word2vec* приймає великий текстовий корпус даних у якості входу і продукує векторний простір. Кожне слово з корпусу отримує свій відповідний вектор у цьому просторі. Відмінна риса полягає в тому, що слова зі схожих контекстів у корпусі розташовується у векторному просторі близько одне від одного.

2.4. Порівняння бібліотек для NLP

У *Python* є величезна кількість бібліотек, які можуть бути використані для задачі NLP. Далі наведені плюси та мінуси найбільш популярних з них.

NLTK (Natural Language ToolKit)

+ найбільш відома та повна NLP бібліотека;

+ багато розширень, розроблених третьою стороною;

+ багато підходів для кожної NLP задачі;

+ швидка токенізація (розділення тексту на окремі речення);
+ підтримує найбільшу кількість мов у порівнянні з іншими бібліотеками.

– доволі складна для вивчення та використання;
– у цілому, доволі повільна для повного циклу задач NLP;
– під час токенізації, NLTK розділяє текст на речення не аналізуючи семантичну структуру;
– базується на обробці рядків, а не об’єктів, що є не дуже звичним підходом у *Python*;
– не надає моделей нейронних мереж;
– відсутність інтегрованих словесних векторів.

spaCy

+ найшвидша NLP бібліотека;
+ легка для освоєння та використання, адже вона має єдиний але дуже оптимізований інструмент для кожної задачі;
+ під час роботи розробник маніпулює об’єктами – більш об’єктно-орієнтована відносно інших бібліотек;
+ використовує нейронні мережі для тренування певного набору моделей;
+ вбудована підтримка словесних векторів;
+ активна підтримка та розроблення станом на 2019 рік.

– не надає такої гнучкості як *NLTK*;
– токенізація речень менш повільна, у порівнянні з *NLTK*;
– не підтримує велику кількість мов. Станом на весну 2019 року, існують моделі для 7 мов і одна “мультимовна” модель.

DeepPavlov

+ спрямованість саме на розробку чат-ботів та комплексних розмовних систем (у тому числі NLP-систем та діалогових систем);
+ бібліотека надає можливість використання великої кількості компонентів для вирішення різних підзадач NLP, таких як NER, морфологічне позначення, а також настроювання параметрів моделей;
+ легка у використанні, адже доступ до складних компонентів надається через інтерфейси зі зрозумілими найменуваннями;
+ зрозуміла офіційна документація, яка покриває абсолютно усі компоненти системи, які надаються для використання користувачам бібліотеки;
+ побудована з використанням провідних бібліотек *Keras* та *TensorFlow*.

– для тренування власних моделей потрібні доволі великі обсяги оперативної пам’яті (мінімум 16 ГБ);
– так як бібліотека – нова і активно розвивається станом на 2019 рік, кожне оновлення бібліотеки, навіть незначне, може потягнути за собою значні зміни у архітектурі системи, і, відповідно, потреби у зміні коду користувачів бібліотеки.

scikit-learn

+ надає широкий вибір алгоритмів для будування алгоритмів машинного навчання;
+ має функції, які допомогають використовувати модель під назвою “мішок слів” (bag-of-words, bow model) для створення функцій для проблем класифікації тексту;

+ має якісну документацію та інтуїтивно зрозумілі назви методів та класів бібліотеки.

– для більш складної обробки даних – наприклад, для визначення частин мови (POS-tagging – part-of-speech tagging) – потрібно використовувати сторонню NLP бібліотеку, і тільки після цього використовувати моделі з даної бібліотеки;
– не використовує нейронні мережі для обробки тексту.

gensim

+ добре справляється з великими наборами даних та підтримує "живі" потоки даних;
+ надає tf-idf (term frequency – inverse document frequency) векторизацію (оцінка важливості слова у контексті), word2vec, document2vec, латентний семантичний аналіз, латентне розміщення Діріхлє;
+ наявна підтримка алгоритмів глибокого навчання.

– першочергово розроблена для текстового моделювання "без нагляду" (unsupervised);
– не надає достатньої кількості інструментів для повного циклу задач NLP, і, таким чином, має використовуватись з певною іншою бібліотекою (*spaCy* або *NLTK*).

Pattern

+ підтримує визначення частин мови слів, пошук по n-граммам, сентиментальний аналіз, WordNet, векторну модель, кластеризацію та метод опорних векторів;

+ включає в себе “веб-сканер” (web crawler), DOM-парсер та підтримку деяких API (напр., Twitter, Facebook та інші).

– бібліотека першочергово розроблена для збору та аналізу контенту веб-сторінок, і не може бути оптимізована для ряду NLP задач.

Polyglot

+ підтримує величезну кількість мов (від 16 до 196 мов в залежності від типу NLP задачі).

– бібліотека не є дуже популярною, як, наприклад, *NLTK* або *spaCy*, і може надавати повільніші результати за інші бібліотеки;
– слабка підтримка з боку спільноти бібліотеки.

2.5. Результати аналізу бібліотек

Базуючись на порівнянні усіх приведених бібліотек, прийняте рішення використовувати бібліотеку *DeepPavlov*. Дана бібліотека є найбільш релевантною для задачі дипломного проекту, який ставить за основну мету створення *ефективного* програмного застосунку.

DeepPavlov – це бібліотека для вирішення NLP задач. Як вказано у офіційній документації [19], вона є більш націленою саме на вирішення задачі побудови систем питання-відповіді та пов’язаних з нею NLP підзадач.

Бібліотека надає доступ до таких речей як:

- набір попередньо натренованих NLP моделей, попередньо визначених компонентів діалогових системи (базованих на ML, DL або визначених правилах) та шаблонів рішення різних задач NLP;
- фреймворк для створення та тестування власних діалогових моделей;

- інструменти для інтеграції безпосереднього NLP рішення з суміжною інфраструктурою (месенджери та ін.)
- оточення для оцінки розмовних моделей та доступ до релевантних наборів даних.

Бібліотека має наступні ключові концепції:

- *Agent (Агент)* – це розмовний агент, який спілкується з користувачами природньою мовою;
- *Skill (Вміння)* – задовольняє ціль кінцевого користувача у розрізі певної задачі. Зазвичай, це досягається через надання інформації чи виконання певної транзакції (наприклад, відповідь на часто задаване питання, бронювання білетів і таке інше). Однак, для певного виду задач успішна інтеракція досягається за допомогою тривалого спілкування (як людська розмова)
- *Component (Компонент)* – функціональна частина Вміння, яку можна використовувати багаторазово;
- *Rule-based Models (Моделі, які базуються на визначених правилах)* не можуть бути натреновані.
- *ML Models (моделі ML)* можуть бути натреновані тільки самостійно, без залежності від інших моделей;
- *DL Models (моделі DL)* можуть бути натреновані незалежно від інших моделей, а в натренованому вигляді їх використання може бути об’єднано у ланцюжок;
- *Skill Manager (Менеджер Вмінь)* виконує вибір Вмінь для створення відповіді;
- *Chainer (Поєднувач)* будує агента (чи шаблон використань компонентів) з неоднорідних компонентів (ML, DL або моделей, які базуються на визначених правилах). Це дозволяє об’єднувати декілька моделей у єдине ціле для тренування та використання.

Найменша частина бібліотеки, з якої починається будова рішення, - це Компонент. Компонент – це будь-яка функція, яка виконує певну під задачу NLP. Він може бути імплементований у виді нейронної мережі, ML моделі або моделі, яка базується на визначених правилах. Окрім цього, Компонент може мати вкладену структуру, тобто Компонент може включати в себе інші Компоненти.

Компоненти можуть бути об’єднані у Вміння. Вміння вирішує більш об’ємну задачу NLP, порівнюючи з Компонентом. Однак, у розрізі імплементації, Вміння не сильно відрізняються від Компонентів. Єдине обмеження Вмінь у тому, що їх вхідні и вихідні дані мають бути рядками. Таким чином, Вміння часто пов’язані з окремими завданнями діалога.

Використання Агента передбачається у вигляді багатоцільової діалогової системи, яка включає в себе декілька Вмінь, і може переключатися між ними. Це може бути діалогова система, яка має у собі Вміння для ведення розмови та Вміння, яке націлене на певний результат (наприклад, бронювання квитка), і обирає одне з них для створення відповіді, базуючись на вхідних даних від користувача.

DeepPavlov використовує можливості бібліотек *TensorFlow* та *Keras*.

TensorFlow – це бібліотека, створена і підтримувана компанією *Google*. За допомогою даної бібліотеки можна ефективно розроблювати та тренувати моделі машинного навчання.

Keras – бібліотека, яка надає високорівневий *API* до нейронних мереж. Вона дозволяє легко і швидко створювати прототипи нейронних мереж завдяки своїй зручності, модульності та розширюваності. Бібліотека підтримує як згорткові мережі, так і рекурентні мережі, а також комбінації обох. Бібліотеку можна використовувати як на базі звичайного процесора, так і на базі відеокарти (графічного процесора). Окрім цього, бібліотека має нативну підтримку бібліотеки *TensorFlow*.

2.6. СКБД

Дана система передбачає використання СКБД. База даних має бути сховищем для основної інформаційної бази, у якій буде першочергово проводитися пошук відповіді на запит – тобто зберігати визначені часто задавані питання та відповіді на них.

Початковий об’єм даних – невеликий, однак передбачається, що список питань буде розширюватися, і туди будуть додаватися нові питання, які, можливо, були неактуальними або раніше не задавалися.

Перш за все, необхідно обрати модель даних та відповідний вид СКБД: реляційну (*PostgreSQL*, *MySQL*, *Microsoft SQL Server*) або нереляційну (*MongoDB*, *Redis*).

Розробником обрана нереляційна модель даних. Дана модель добре підходить під задачі, які передбачають роботу з неструктурованими даними – такими, у яких нема конкретної визначеної схеми.

Серед наведених нереляційних СКБД обрано *MongoDB*. Незважаючи на те, що *Redis* – це невелика та швидка база даних, яка досягає рекордних швидкостей доступу за рахунок того, що вона зберігається у оперативній пам’яті, у неї є серйозний недолік – система забезпечує швидкість роботи за рахунок часу зберігання даних: якщо сервер, на якому працює дана база даних, вимкнути або перезавантажити, усі дані буде втрачено. Таким чином, *Redis* – хороший вибір для зберігання виключно актуальних даних, які не представляють постійної цінності, і до яких може знадобитися швидкий доступ – наприклад, поточні ціни на фондових біржах.

Дані в *MongoDB*, на відміну від *Redis*, – зберігаються на файловій системі, тому при вимиканні серверу, на якому працює СКБД, або збоях у його роботі (за виключенням проблем безпосередньо з накопичувачами даних) дані не знищуються і до них можна буде отримати доступ. Властивість постійності даних, які зберігаються, важлива для даної системи, і тому, серед двох наведених нереляційних СКБД, обрано *MongoDB*.

3. ОГЛЯД РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

3.1. Структура програмного застосунку та бази даних

Структура програмного застосунку

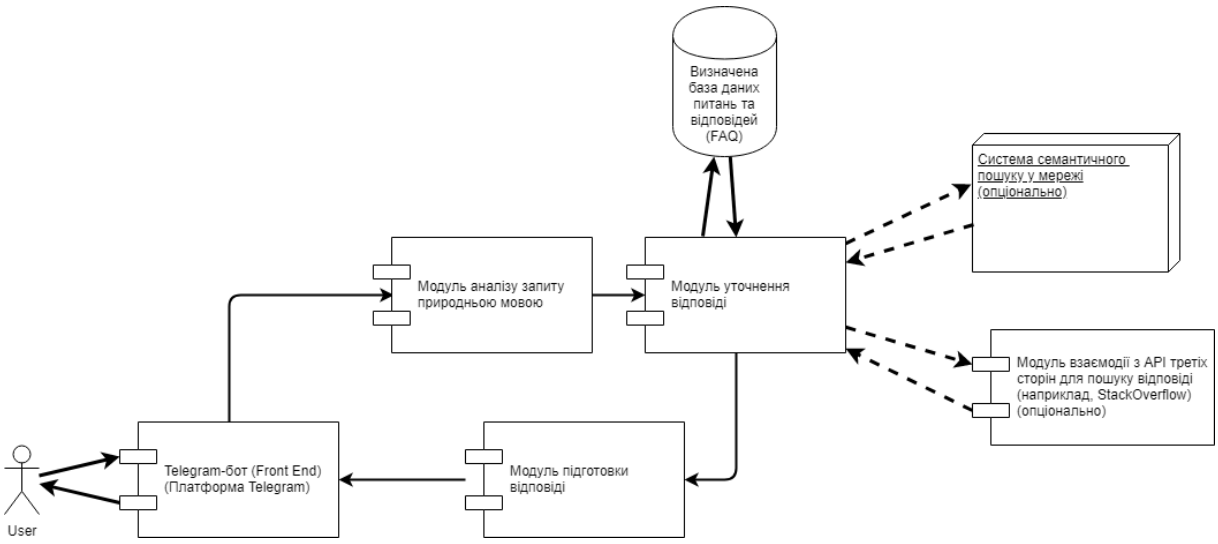


Рис. 3. Архітектура розробленої системи

Оглянемо основні структурні компоненти (модулі) програмного застосунку на прикладі спрощеної структурної схеми. Повна схема наведена у Додатках.

- *Telegram-бот*. Це точка взаємодії між користувачем та системою. Взаємодія відбувається через бота, розміщеного на платформі *Telegram*, через задання питання користувачем у інтерфейсі чату, і, після обробки питання системою і підготовки відповіді, отримання відповіді у межах цього ж чату;
- *Модуль аналізу запиту природньою мовою.* Це проміжний модуль між модулем, який отримує питання від користувача, та модулем, який звертається до певних джерел даних, щоб знайти відповідь на це питання. У даному модулі виконується оброблення тексту питання та виділення з нього особливих сутностей для питання, таких як ключові слова;

- *Модуль вибору відповіді.* Даний модуль взаємодіє з джерелами даних (див. наступний пункт) з метою пошуку відповіді на поставлене користувачем питання;
- *Джерела даних.* Джерела даних – це компоненти, або навіть окремі системи, у яких та/або за допомогою яких можна знайти відповідь, або декілька відповідей, на поставлене питання. Основним та обов'язковим компонентом є наперед сформована база даних часто задаваних питань та відповідей до них. Дане джерело даних є пріоритетним для пошуку відповіді, адже питання, підібрані у даному джерелі, досить часто зустрічаються, і відповідь на них підготовлена людиною вручну. Така відповідь, скоріш за все, буде більш прийнятною, ніж відповідь, сформована системою автоматично – але така відповідь буде надана, якщо таке питання взагалі буде знайдено у цьому джерелі. Інші джерела даних є опціональними. Вони, за наявності, будуть використані для пошуку відповідей або посилань, які можуть містити відповіді. Прикладами таких компонентів можуть слугувати компоненти, які дозволяють виконувати семантичний пошук у мережі, базуючись на певних ключових словах, виділених з питання, або API певної третьої сторони – наприклад, сайту *StackOverflow* [20], який є платформою, де задають питання, здебільшого по темі програмування, та отримують на них відповідь.
- *Модуль підготовки відповіді.* Даний модуль, базуючись на відповіді, яка надійшла з модуля вибору відповіді, візуально її оформлює, застосовуючи до відповіді певні функції форматування, виділяючи в окремі абзаци текстову частину; посилання, які можуть бути використані для отримання більш ширшої відповіді, окремі фрагменти коду та ін.

Структура бази даних

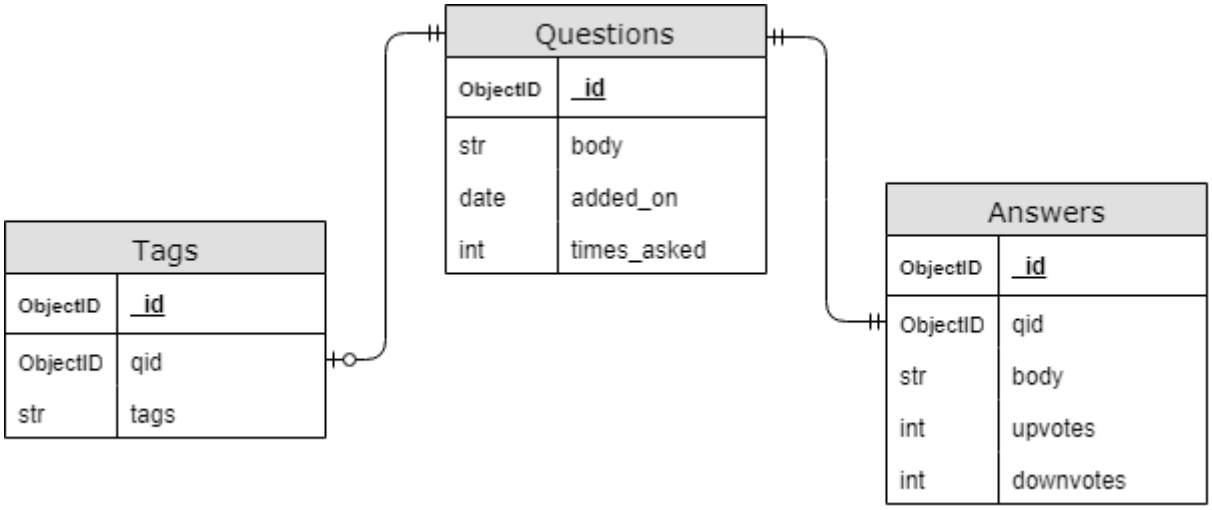


Рис. 4. Схема бази даних

База даних складається з трьох колекцій.

Перша колекція, "*Questions"* (Питання), – це колекція, документами якої є наперед визначені питання, які часто задають студенти.

У сутності "Питання" наявні наступні поля:

- *_id (ObjectId)* – унікальне значення, яке автоматично присвоюється при доданні документа до колекції, та яке слугує ідентифікатором конкретного питання;
- *body (str)* – безпосередньо текст питання;
- *added_on (date)* – дата додання питання до бази даних;
- *times_asked (int)* – кількість разів, коли система, базуючись на вхідних даних, вважала, що задається таке питання.

Друга колекція, *"Answers"* (Відповіді), – це колекція, документами якої є наперед визначені відповіді на згадані вище питання.

У сутності "Відповідь" наявні наступні поля:

- *_id (ObjectId)* – унікальне значення, яке автоматично присвоюється при доданні документа до колекції, та яке слугує ідентифікатором конкретної відповіді;
- *qid (ObjectId)* – поле, яке пов'язує відповідь з конкретним питанням. Є посиланням на документ з колекції *"Questions"*.
- *body (str)* – безпосередньо текст відповіді;
- *upvotes (int)* – кількість разів, коли, при наданні відповіді користувачеві, вона виявилася для нього корисною;
- *downvotes (int)* – кількість разів, коли, при наданні відповіді користувачеві, вона виявилася для нього некорисною.

Остання колекція, *"Tags"* (Теги), – це колекція, документами якої є певні теги (ключові слова), якими можна характеризувати певне питання.

У сутності "Тег" наявні наступні поля:

- *_id (ObjectId)* – унікальне значення, яке автоматично присвоюється при доданні документа до колекції.
- *qid (ObjectId)* – поле, яке пов'язує тег з конкретним питанням. Є посиланням на документ з колекції *"Questions"*.
- *tag (str)* – безпосередньо сам тег.

3.2. Підхід до аналізу запиту природною мовою

Модуль, який виконує аналіз запиту природною мовою має на меті аналіз текстового представлення питання, яке задається користувачем, видалення з нього нерелевантних для пошуку відповіді слів, та, навпаки, виділення ключових слів, за якими можна було б провести пошук по

визначеній базі даних відповідей або які можна передати у якості вхідних параметрів іншому модулю пошуку відповіді.

Видалення непотрібних у термінах пошуку слів називається процесом вилучення стоп-слів. Стоп-слова – це слова, за допомогою яких будується граматично та логічно правильне речення для сприйняття його людиною, але які є для автоматизованої пошукової системи нічим іншим, як "шумом", який не покращить, а, можливо, й погіршить результати роботи системи. Даний набір слів можна або ігнорувати на кожному етапі алгоритму з аналізу вхідних даних, або, що більш коректно у термінах оптимізації, видалити на одному з кроків обробки вхідних даних, проаналізувавши увесь текст запиту. Ці списки стоп-слів, зазвичай, включають в себе сполучники, такі як *"and"* (і), *"or"* (або), *"but"* (але); слова, які несуть ввічливе забарвлення: *"please"* (будь ласка), *"thanks"* (спасибі) та ін. Також можливе додання до цього списку інших слів, які часто зустрічаються у конкретній області, адже вони будуть завжди фігурувати серед ключових слів. І, таким чином, не будуть фактично являти собою ключові слова, адже фігуруватимуть майже у всіх питаннях, і пошук по ним не дасть коректних результатів.

Виділення ключових слів, за якими можна провести пошук у відповідних джерелах інформації, полягає у в визначенні слів у вхідному запиті, які формують основу самого питання. Ці окремі слова можуть стати у нагоді при пошуку інформації (відповіді), яка задовольнить користувача системи. На прикладі доволі легкого питання: *"What is integer?"* ("Що таке цілочисельний тип даних?"), ключовим словом буде *"integer"*. Наведемо приклад більш складного питання: *"I can't understand, my Eclipse crashes when I try to launch it, it crashes 8 out from 10 times I use it, help plsss"* ("Я не розумію, мій Eclipse аварійно завершає свою роботу, коли я намагаюся його запустити, він аварійно завершається у 8 з 10 разів, коли я його використовую, будь ласка, допоможіть"). Тут ключовими словами буде декілька слів, сортовані в порядку пріоритетності: *"crash"*,

“Eclipse” (IDE), “launch”. Слід сказати, що, дійсно, якщо подивитися на даний набір виділених слів, то людина, скоріш за все, зможе сказати, з якою проблемою пов’язане дане питання.

Обидві з описаних підзадач вирішуються за допомогою розрахунку та використання величини *TF-IDF*. За допомогою величини *TF-IDF* характеризується кожне конкретне слово. Даний показник прямо пропорційний частоті вживанню даного конкретного слова, виділеного з вхідного запиту, до частоти вживань цього слова у всіх інших запитах (документах), які використовувалися для тренування моделі, яка проводить підрахунок даної величини:

$$tf\text{–}idf(t,d,D) = tf(t,d) * idf(t,D), \tag{1}$$

де

$$tf(t,d) = \frac{n_t}{\sum_k n_k}, \tag{2}$$

де d – документ, n_t є числом входжень слова t, сума у знаменнику – це загальна кількість слів у документі, а

$$idf(t,D) = \log \frac{|D|}{\left|\{ d_i \in D \mid t \in d_i \}\right|}, \tag{3}$$

де $|D|$ – кількість документів у колекції, а $\left|\{ d_i \in D \mid t \in d_i \}\right|$ – число документів з колекції, у яких зустрічається t (коли $n_t \neq 0$).

Для того, щоб натренувати модель, яка змогла б виконувати дані задачі, потрібні певні вхідні дані. Так як тематика вхідних питань має здебільшого бути пов’язаною з програмуванням, набором даних, який обрано розробником для тренування моделі, – є набір питань з популярного сервісу *StackOverflow* [20], де програмісти як початкового, так і більш передового рівня знань щодня задають різні питання.

Користуючись відкритим набором даних від компанії *Google* [21], було випадковим чином обрано 20’000 питань, які будуть слугувати даними для тренування моделі.

Перед тим, як використати дані – заголовки та безпосередньо самі тексти питань, – виконано підготовку (обробку) цих текстових даних. Усі слова були приведені до нижнього регістру, а також були видалені знаки пунктуації, такі як крапка, тире, слеш та інші спеціальні символи. Також, вирішено прибрати усі цифри. Після цього, заголовки і тексти питань об’єднувалися в один рядок.

Наступний крок – це створення словника унікальних слів. Щоб підвищити ефективність другого кроку – виділення ключових слів – з масиву слів, який сформовано зі всіх питань, були вилучені стоп-слова. За базу взятий список стоп-слів [22], куди також були додані стоп-слова, специфічні для вищезгаданого набору даних. *StackOverflow* дозволяє форматувати тексти питань та відповідей, додаючи до них різні стилі (такі як **напівжирний шрифт**, *курсив*, відповідний шрифт для блоків коду) або інші елементи форматування (списки, посилання тощо). Дане форматування реалізується через *HTML*-теги, які у тексті вхідних даних виглядають як: *<b> (lt;bgt;), <i> (lt;igt);* та інші – вони також були додані до списку стоп-слів. Окрім цього, туди були додані деякі слова, які зустрічаються більш ніж у 80% питань, такі як *“project”* (проект) та інші, і які, відповідно, не можна характеризувати як **ключові** слова для конкретного питання.

Після аналізу 20’000 питань, сформований словник вміщав у себе близько 120’000 слів. Слід зазначити, що загальноприйнятою практикою є обмеження кількості слів у словнику. Це є логічним кроком, адже, по-перше, менша кількість слів потребуватиме менших об’ємів пам’яті, щоб використовувати словник, а, по-друге, тим самим ми ще раз виділимо найбільш ключові слова з наявного набору. Таким чином, розмір словника зменшено до перших 10’000 найбільш вживаних слів. Даний взаємозв’язок

документів та словника можна представити через векторну модель: кожному документу буде відповідати свій вектор унікальних слів. Розмірності цих векторів будуть співпадати з розмірністю словника слів, тобто, в даному випадку, 10’000. За кожним словом буде закріплений свій відповідний індекс у кожному з побудованих векторів), а значенням за цим індексом буде кількість разів, коли це слово використано у документі. Виконавши дані розрахунки та перетворення, використовуючи формулу (2), ми отримаємо величину *TF*.

Тепер, для кожного слова слід порахувати величину *IDF*. Якщо величина *TF* характеризувалася унікальною для кожного з слів у межах одного документа, то величина *IDF* буде характеризувати кожне з слів зі словника у межах усього набору документів. Для підрахунку цієї величини слід використати векторну модель, отриману раніше. У даній моделі, якщо слово зустрічається у документі хоча б 1 раз, воно буде мати відрізне від 0 значення у відповідному індексі вектора, інакше – 0. За формулою (3) ми можемо провести підрахунок величини *IDF*.

Тепер, маючи значення TF для кожного слова з певного документа та IDF для кожного слова, можна порахувати величину *TF-IDF* за формулою (1) у межах конкретного документу, або запиту. Підраховуючи величину *TF* для конкретного запиту, та, застосовуючи пораховану на наборі даних *StackOverflow* величину *IDF*, ми отримаємо результати для конкретних слів, які, відсортовані за спаданням, будуть представляти список ключових слів для наданого питання.

3.3. Модуль вибору відповіді

Модуль, який отримує на вхід результати модулю, який виконує аналіз запиту природною мовою та виділяє з нього важливий контекст.

Модуль має таку назву, тому що, насамперед, дана робота передбачає використання єдиного обов’язкового джерела даних – це визначеної наперед бази даних питань та відповідей. Однак, архітектура

даного застосунку також дозволяє додавати інші джерела даних або застосовувати інші підходи до пошуку інформації, тим самим, доповнюючи той набір інформації, який міститься у основній базі даних.

Отже, першочергово, за наданим модулю контекстом (ключовими словами) відбувається пошук у базі даних, описаній у розділіі 3.1. Якщо у базі даних знаходиться питання, яке, за розрахунками системи, вважалося би відповідним або схожим до запиту, який задав користувач, то користувачу надається пов’язана з цим питанням відповідь, а відповідне поле питання у базі даних, *times_asked,* збільшується на одиницю. Інакше, модуль вибору відповіді передає результати аналізу до інших опціональних модулів пошуку відповіді, такі як модуль семантичного пошуку по мережі або API третіх осіб), та, отримуючи від них повноцінну відповідь або окремі частки (параграфи, абзаци, посилання), з яких можна сформувати відповідь, передає їх наступному модулю системи – модулю підготовки відповіді.

3.4. Модуль підготовки відповіді

Модуль підготовки відповіді – це модуль, який слугує для того, щоб на основі результатів, отриманих від попереднього модулю вибору відповіді, сформувати належним чином відповідь, яка була б не тільки інформативною, але й читабельною.

Цей модуль є останнім кроком перед фінальним наданням користувачу відповіді. Не дивлячись на те, що даним модулем можна знехтувати, адже на даному етапі, вхідні дані вже несуть у собі, за розрахунками системи, результат, який можна надати користувачу, та останній зможе виділити з нього інформацію, що його цікавить. Але користувач отримає більше задоволення від користування системою, якщо вона, хоча б на певному рівні, спробує структурувати відповідь та відсортувати інформацію, надану в неї, за релевантністю. Цей крок, якщо навіть виконаний на базовому рівні, вже стане у нагоді для того, хто

ставить запитання, адже він витратить менше часу на розуміння відповіді, або, якщо надається розкрита відповідь, першочергово отримає найбільш важливу інформацію, яку змогла знайти система.

Отже, даний модуль несе у собі ряд функцій, що відносяться до форматування. У даному модулі розділяються текстова частина відповіді та доповнення до неї, такі як посилання (наприклад, посилання, за яким знайдено відповідь, або посилання, за яким можна знайти більшу кількість інформації відносно поставленого питання), або навіть фрагменти коду, якщо такі надаються альтернативними модулями надання відповідей. Також, тут можуть застосовуватися форматування для окремих речень чи слів – **напівжирний шрифт**, *курсив*.

4. АНАЛІЗ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

4.1. Опис інтерфейсу

Інтерфейс даної дипломної роботи виконаний у вигляді діалогу з ботом, розміщеного на платформі *Telegram*. Безпосередньо графічну частину надає сам сервіс, логіка ж взаємодії з цим інтерфейсом описується розробником окремо.

Взаємодія з ботом відбувається таким же чином, як і користування будь-яким іншим приватним (тобто у якому задіяні два співрозмовника) чатом месенджера – за виключенням того, що повідомлення, які будуть надсилатися другою стороною, будуть згенеровані автоматично, та які, скоріш за все, будуть надсилатися з певною затримкою, адже системі потрібно буде обробити питання, знайти на нього відповідь, і тільки опісля надати користувачеві.

У найпростішому варіанті взаємодія з ботом відбувається наступним чином: користувач пише в чат питання, а бот, після обробки питання і знаходження на нього відповіді, надає її (див. рис. 5).

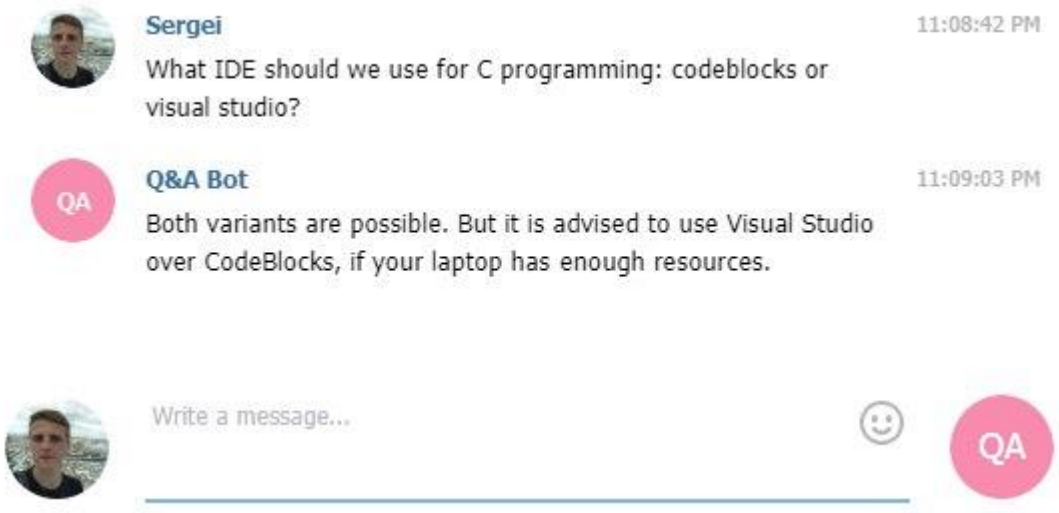


Рис. 5. Діалог з Telegram-ботом

Якщо ж у бота не вдалося знайти відповіді на поставлене запитання, у якості відповіді використається зарезервоване під такі ситуації повідомлення (рис. 6).

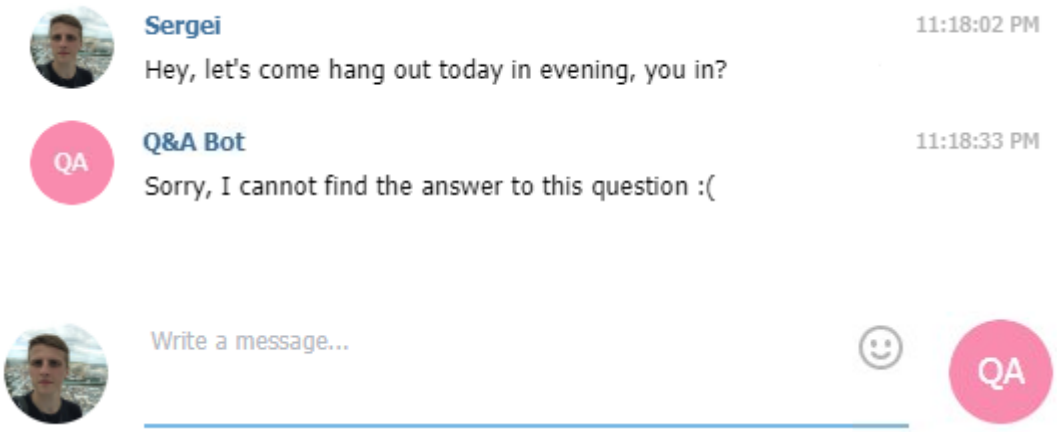


Рис. 6. Резервна відповідь

4.2. Тестування системи

Як й будь-яке ПЗ, яке пройшло основний етап розробки і дійшло до етапу релізу, – дана система має бути ретельно протестована. Коли ПЗ доступне кінцевим користувачам, слід бути впевненим у тому, що система функціонує, причому функції, доступ до яких вона надає, – працюють коректно.

Для тестування системи були використані різні підходи з тестування. Серед них – димове тестування та функціональне тестування.

Димове тестування – підхід, який має на меті поверхневе тестування більшості (або навіть всіх) компонентів системи. Даний вид тестування передбачає написання невеликого числа тестів, або помірної кількості невеликих та нескладних тестів, набір з яких дозволить довести (або, навпаки, заперечити) працездатність системи “в цілому”, витративши на це – як написання тестів, так і їх проведення – незначну кількість часу.

Прикладом такого тестування у межах системи може слугувати досить очевидний сценарій тестування та його очікуваний результат:

наведення питання боту та очікування на це питання відповіді. Якщо відповідь надійшла – значить, система, в цілому, працює. Також, слід дещо більше протестувати систему, задавши декілька питань, кожне з яких слід орієнтувати на тестування окремих джерел даних. Таким чином, якщо наявне одне або більше з опціональних джерел, слід задати таке питання, яке не буде знайдене у наперед визначеній (основній) базі даних, – для того, щоб активізувати виконання частини системи, пов’язаної з роботою з вторинними джерелами даних. При цьому, не слід оцінювати якість надаваної відповіді – це не входить в мету димового тестування, і даний параметр оцінено під час інших тестувань, наприклад, функціонального.

Функціональне тестування – це тестування, яке навпаки направлене на ретельну перевірку окремої функції системи. Прикладом функціонального тестування є підбір різних, звичайних та граничних, валідних та неочікуваних вхідних параметрів, та передача цього набору параметрів на вхід до тестованої функції.

Функціональне тестування у межах системи можна описати на прикладі тестування двох функцій з модуля аналізу запиту природної мови: функція, яка “очищає” запит від нерелевантних символів, і функція, яка виділяє з нього ключові слова.

Для функції очистки запиту набори вхідних даних (тобто користувацькі запити – рядки) слід подавати з символами у різному регістрі, та очікувати однаковий результат; також, результат слід перевіряти на те, що в ньому відсутні стоп-слова, які мали видалитися після обробки; передавати граничні значення (такі як пустий рядок) та інше.

На функцію виділення ключових слів слід також подавати граничні значення у вигляді пустого рядка. Результат, який отримується на будь-якому наборі вхідних параметрів, можна перевіряти на те, щоб кожне слово з виділених ключових слів було дійсно наявно у вхідному запиті – інакше, дане слово отримано певним іншим чином у функції, що є

некоректним, адже вона не виконала описані для неї функціональні вимоги – виділити ключові слова саме з запиту.

Слід також зазначити те, що тестування зручності користування як таке не проводилося у межах даної системи. Так як було прийняте рішення про розміщення системи у якості бота на платформі *Telegram*, то зручність користування ботом в основному залежить від того, наскільки доступним є графічна складова інтерфейсу взаємодії з ботом. Графічна частина, як згадувалося раніше, - повністю розробляється безпосередньо командою месенджера, і на її відображення у межах розробки поточної системи вплинути неможливо – ця складова не є прямою частиною застосунку, що розробляється у межах роботи. Однак, так як месенджер є популярним, як описано у другому розділі даної роботи, а інтерфейс месенджера – комфортність використання, візуальна складова, – є одною з найважливіших складових оцінки месенджера. І, якщо ним користуються, можна зробити висновок про те, що ці складові виконані на високому рівні.

4.3. Рекомендації до користування додатком

Функції даної системи є доступними при будь-яких вхідних параметрах. Але системі не завжди вдасться задовольнити користувача та підібрати для нього найбільш релевантну відповідь на його питання – через ряд особливостей під час формування тексту запиту.

Система має на меті спрощення взаємодії з людиною – також завдяки тому, що людина може формувати свої питання у вигляді, наче вона задає їх іншій людині (тобто їй не потрібно змінювати звичний для неї підхід). Але, в такому випадку, застосунок поки що не здатний показати кращі результати за людину, яка має досвід у тематиці задаваних питань.

З іншої сторони, щоб спробувати отримувати більш релевантні відповіді від системи, можна спробувати формувати запити для системи як для певної машини, враховуючи її особливості.

Так, якщо бот є мономовним, не слід задавати питання на інших відмінних мовах, або використовувати окремі слова з іншої мови – це, в кращому випадку, не дасть впливу на результати роботи, а, в гіршому, послугує “шумом” для системи.

Також, слід зазначити, що пунктуаційні знаки відфільтровуються на етапі аналізу запиту. Окрім цього, не слід використовувати у запитах спеціальні символи, адже, якщо вони не будуть відфільтровані, вони також можуть послугувати “шумом”.

Окрім цього, слід уточнити, що замість того, щоб використовувати слова у відповідній ним формі, що було б граматично правильно, для даної програмної системи, як і зазвичай для подібних систем, краще подавати слова у інфінітивній формі.

В цілому, треба просто пам’ятати, що це саме програмна система, а не людина, і, при взаємодії з нею, слід застосовувати навички, набуті при взаємодії з іншими схожими системами.

ВИСНОВКИ

Метою даного дипломного проекту є розроблення системи для аналізу запитів, заданих природною мовою.

Проведений у рамках даного проекту аналіз показав, що така система має для виконання своєї прямої функції – аналізу запиту – також підраховувати показник *TF-IDF* для вхідних даних, використовуючи відповідну натреновану модель, та мати достатньо гнучку архітектуру, щоб мати змогу легко розширятися при доданні нових джерел отримання відповіді.

Розроблена автоматизована система:

- проводить обробку природної мови, якою формується запит-питання;
- шукає на нього відповідь наявними у системі модулями пошуку відповідей, насамперед – у наперед визначеній базі даних;
- форматує належним чином знайдені модулями системи відповіді;
- надає відповідь через діалоговий інтерфейс чату на платформі *Telegram*.

Особливу увагу у ході розроблення даного застосунку приділено підготовці широкої бази даних питань та відповідей та модулю аналізу запитів, заданих природною мовою.

Розробку виконано у повному обсязі. Усі вимоги, наведенні у технічному завданні, виконані. Система, згідно з затвердженою методикою виконання, успішно протестована.

Використання даної системи дозволить зменшити час, який витрачає викладач на монотонну роботу, і допоможе студентам швидше знаходити відповіді на свої питання.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Slack – Where Work Happens [Електронний ресурс]. — Режим доступу: https://slack.com/intl/en-ua/
2. Jill Watson Doesn’t Care if You’re Pregnant: Grounding AI Ethics in Empirical Studies [Електронний ресурс]. — Режим доступу: http://www.aies-conference.com/wp-content/papers/main/AIES_2018_paper_104.pdf
3. A teaching assistant named Jill Watson [Електронний ресурс]. — Режим доступу: https://youtu.be/WbCguICyfTA?t=645
4. A teaching assistant named Jill Watson [Електронний ресурс]. — Режим доступу: https://youtu.be/WbCguICyfTA?t=330
5. Viber начнет брать деньги за отправку сообщений – Информатор [Електронний ресурс]. — Режим доступу: https://tech.informator.ua/2019/02/24/viber-nachnet-brat-dengi-s-nekotoryh-polzovatelej-za-otpravku-soobshhenij/
6. Messengers API Comparison – StackShare [Електронний ресурс]. — Режим доступу: https://stackshare.io/stackups/messenger-platform-vs-telegram-bot-api-vs-twilio-api-for-whatsapp
7. Chatbots on Facebook and Telegram: the good and the bad – Tech In Asia [Електронний ресурс]. — Режим доступу: https://www.techinasia.com/chatbots-best-worst-facebook-telegram
8. Java – Wikipedia [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Java_(programming_language)
9. Python – Wikipedia [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Python_(programming_language)
10. Kim, Y. Convolutional Neural Networks for Sentence Classification [Електронний ресурс] / Yoon Kim. — New York

University, 2014. — 6 p. Режим доступу: https://www.aclweb.org/anthology/D14-1181

11. LeCun, Y. Text Understanding from Scratch [Електронний ресурс] / Yann LeCun, Xiang Zhang. — New York University, 2016. — 10 p. Режим доступу: https://arxiv.org/pdf/1502.01710.pdf
12. Тарасов, Д. С. Генерация естественного языка, парафраз и автоматическое обобщение отзывов пользователей с помощью рекуррентных нейронных сетей [Електронний ресурс] / Д. С. Тарасов. — ReviewDot Research, 2015. — 8 с. Режим доступу: http://www.meanotek.ru/files/TarasovDS%282%292015-Dialogue.pdf
13. Lai, S. Recurrent Convolutional Neural Networks for Text Classification [Електронний ресурс] / Siwei Lai, Liu Kang, Liheng Xu, Jun Zhao. — Institute of Automation, Chinese Academy of Sciences, 2015. — 7 p. Режим доступу: https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9745/9552
14. Beaufays, F. Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling [Електронний ресурс] / Francoise Beaufays, Hasim Sak, Andrew Senior. — Google, 2014. — 5 p. Режим доступу: https://wiki.inf.ed.ac.uk/twiki/pub/CSTR/ListenTerm1201415/sak2.pdf
15. He L. Part-of-Speech Tagging [Електронний ресурс] / Lei He, Frank K. Soong, Peilu Wang, Hai Zhao. — Jiao Tong University; Microsoft Research Asian; Educational Testing Service Research, 2015. — 6 p. Режим доступу: https://arxiv.org/pdf/1510.06168.pdf
16. Quoc, V. L. Sequence to Sequence Learning with Neural Networks [Електронний ресурс] / V. Le Quoc, Ilya Sutskever, Oriol Vinyals. —

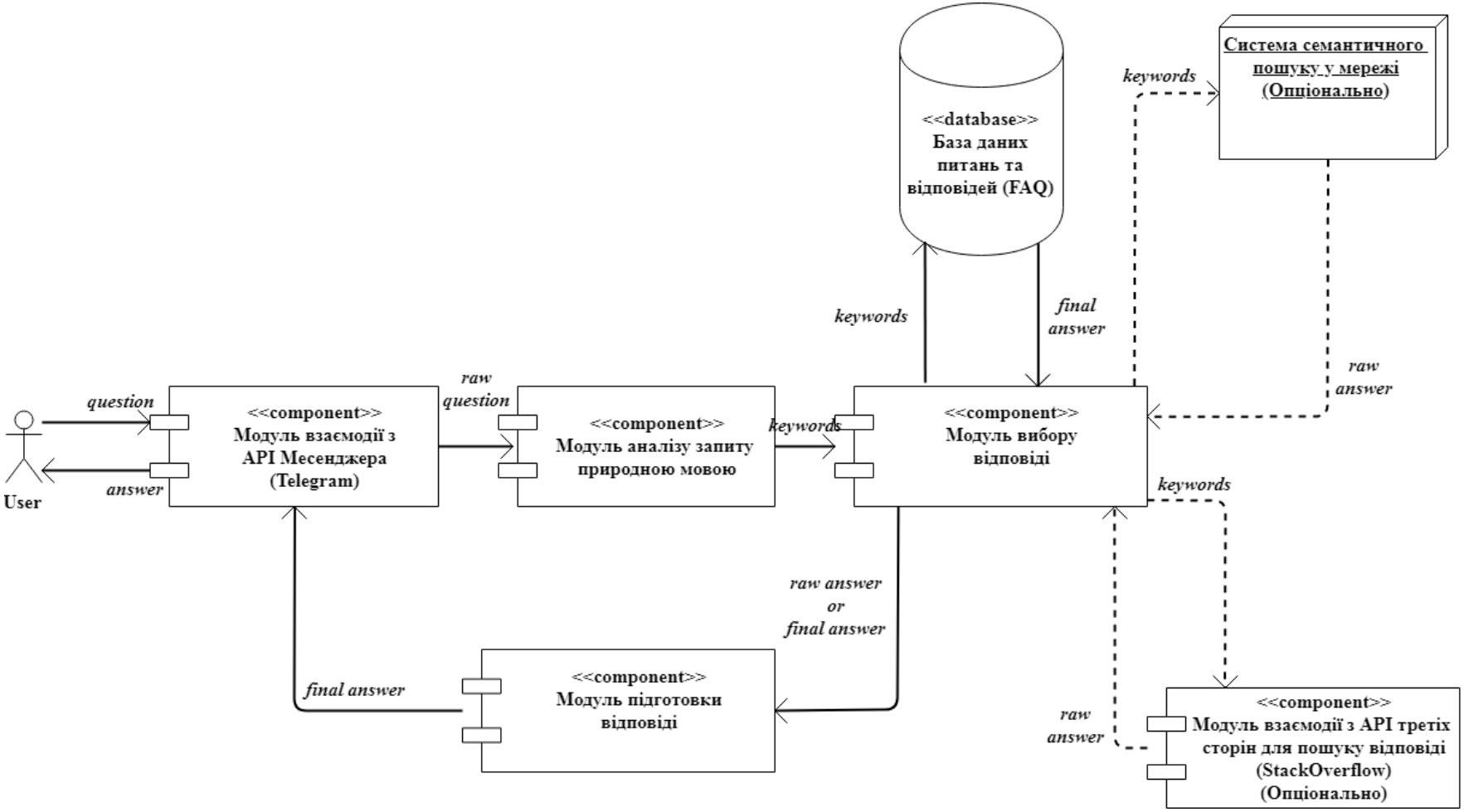
Google, 2014. — 9 p. Режим доступу: https://arxiv.org/pdf/1409.3215.pdf

17. Huang, E. Paraphrase Detection Using Recursive Autoencoder [Електронний ресурс] / Eric Huang. — Stanford University, 2011. — 8 с. Режим доступу: https://nlp.stanford.edu/courses/cs224n/2011/reports/ehhuang.pdf
18. Chen, K. Efficient Estimation of Word Representations in Vector Space [Електронний ресурс] / Kai Chen, Greg Corrado, Jeffrey Dean, Tomas Mikolov. — Google, 2013. — 12 p. Режим доступу: https://arxiv.org/pdf/1301.3781.pdf
19. Welcome to DeepPavlov's Documantation! [Електронний ресурс]. — Режим доступу: http://docs.deeppavlov.ai/en/latest/index.html
20. Stack Overflow - Where Developers Learn, Share, & Build Careers [Електронний ресурс]. — Режим доступу: https://stackoverflow.com/
21. BigQuery Public Datasets [Електронний ресурс]. — Режим доступу: https://cloud.google.com/bigquery/public-data/
22. Stopwords [Електронний ресурс]. — Режим доступу: https://www.ranks.nl/stopwords

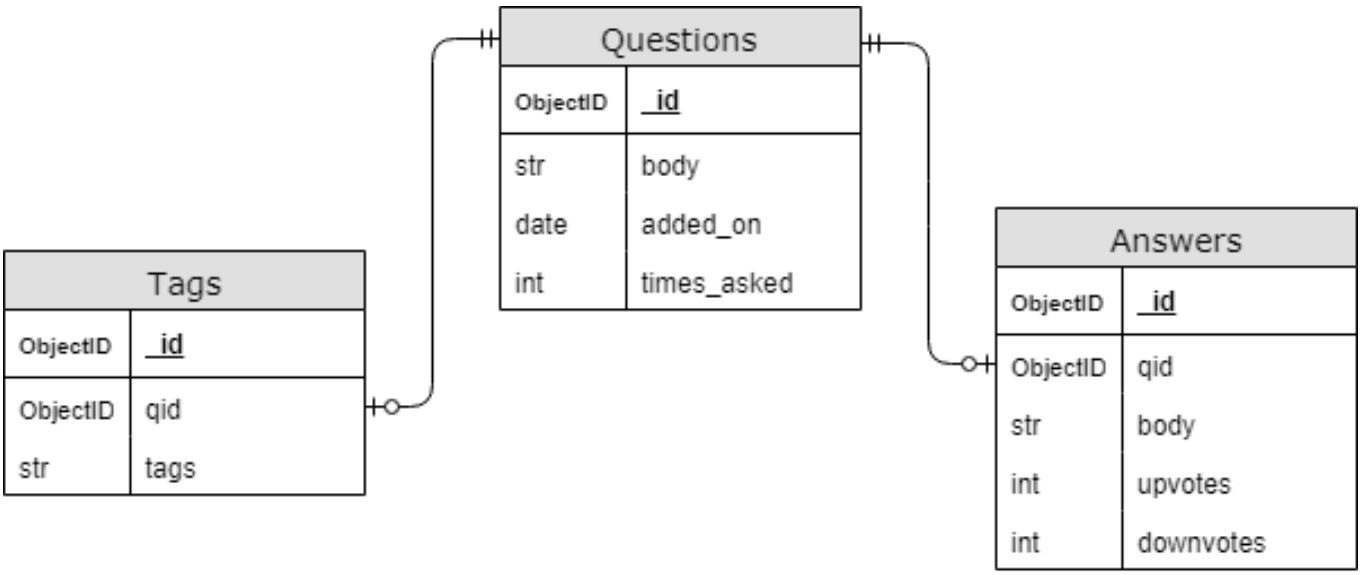
ДОДАТКИ

Додаток 1

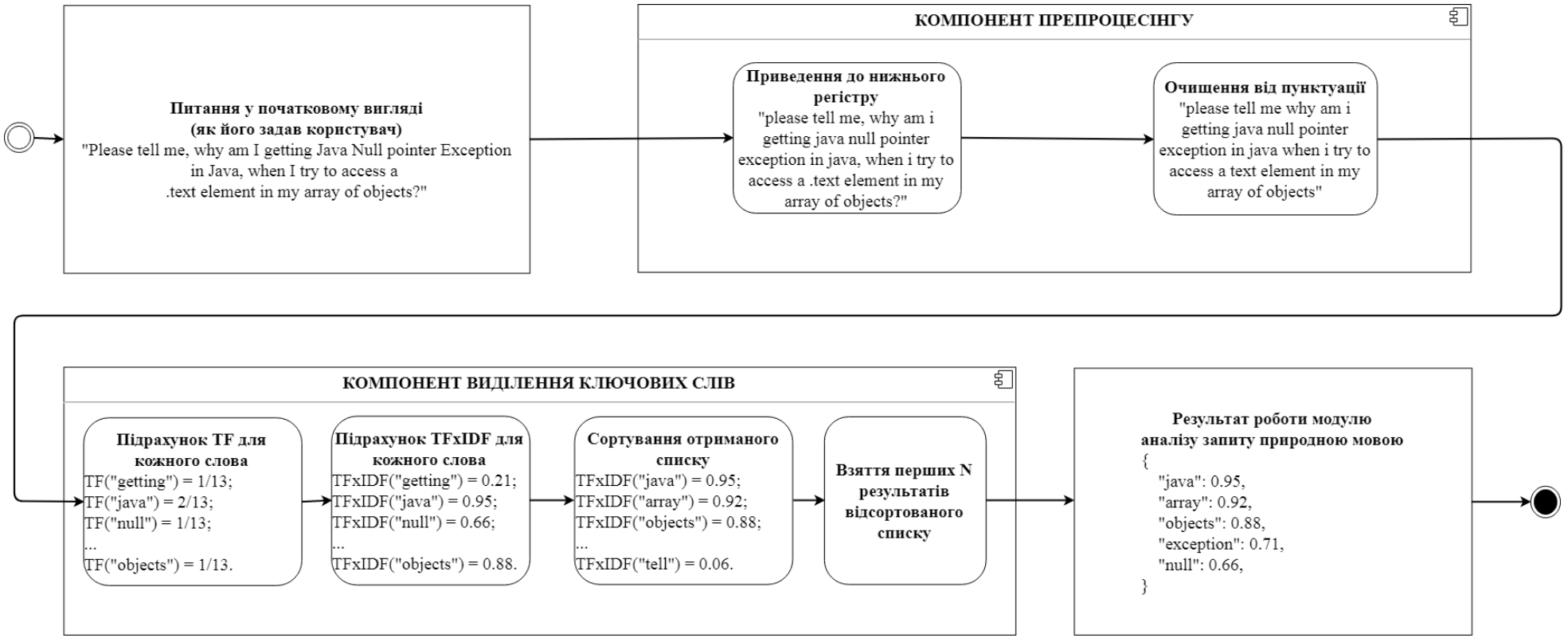
Копії графічних матеріалів



ДП.045440-06-99
Програмна система для аналізу запитів природною мовою. Структурна схема програмної системи. UML діаграма компонентів.

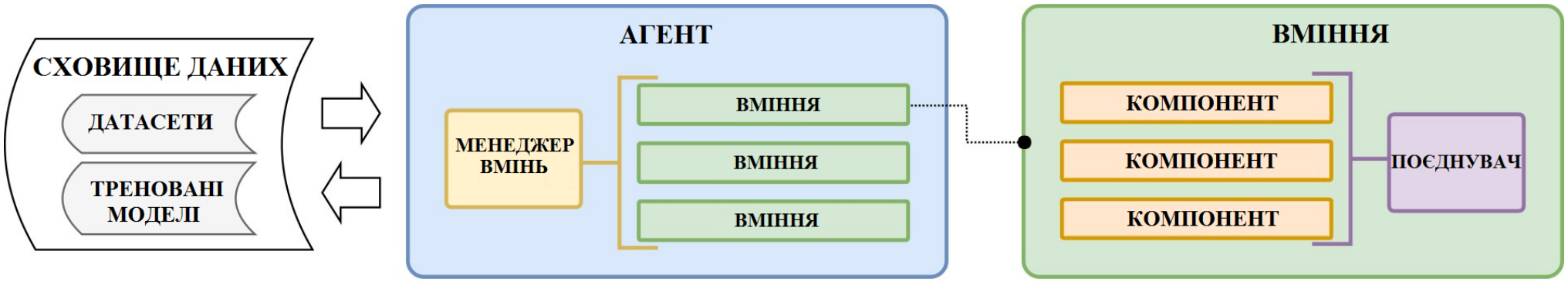


ДП.045440-07-99
Програмна система для аналізу запитів природною мовою. Структура бази даних. ER діаграма.



Алгоритм побудови структурованого питання

Романюк С.О., група КП-51



Особливості архітектури модуля для роботи з базою даних питань та відповідей

Романюк С.О., група КП-51

Додаток 2

Лістинг програми

Модуль виділення ключових слів

```
import re


from singleton import Singleton

from timer import timer


import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer


class KeywordsExtractor(metaclass=Singleton):

    def __init__(self):

        self._tfidf_transformer = None

        self._cv = None

        self._feature_names = None

        self._prepare()


    @timer

    def _prepare(self):

        """

        Prepares TFxIDF model to work with.

        """

        df_idf = pd.read_json("./data/stackoverflow-data-idf.json",
lines=True)

        df_idf['text'] = df_idf['title'] + df_idf['body']

        df_idf['text'] = df_idf['text'].apply(lambda x: self._preprocess(x))

        docs = df_idf['text'].tolist()


        stopwords = self._read_stopwords()

        self._cv = CountVectorizer(max_df=0.85, stop_words=stopwords,
max_features=10000)


        self._tfidf_transformer = TfidfTransformer(smooth_idf=True,
use_idf=True)
```

```
        word_count_vector = self._cv.fit_transform(docs)
        self._tfidf_transformer.fit(word_count_vector)
        self._feature_names = self._cv.get_feature_names()

    def extract_keywords(self, question, max_count=5):
        """
        :param question:
            Question in plain text.
        :param max_count:
            Maximum number of keywords to extract.

            Default to 5.
        :return:
            Dictionary with keywords as keys and their ranks as values, sorted
by ranks.
        """
        tf_idf_vector =
self._tfidf_transformer.transform(self._cv.transform([question]))
        sorted_items = self._sort_coo(tf_idf_vector.tocoo())
        keywords = self._extract_topn_from_vector(sorted_items, max_count)
        return keywords

    @staticmethod
    def _preprocess(text):
        """
        Helper for _prepare()
        """
        text = text.lower()
        text = re.sub("</?.*?>", " <> ", text)
        text = re.sub("(\\d|\\W)+", " ", text)
        return text

    @staticmethod
    def _read_stopwords():
```

```
    """
    Helper for _prepare()
    """
    with open('./data/stopwords.txt', 'r', encoding="utf-8") as f:
        stopwords = f.readlines()
        stop_set = set(m.strip() for m in stopwords)
        return frozenset(stop_set)

@staticmethod
def _sort_coo(coo_matrix):
    """
    Helper for extract_keywords()
    """
    tuples = zip(coo_matrix.col, coo_matrix.data)
    return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True)

def _extract_topn_from_vector(self, sorted_items, max_count):
    """
    Helper for extract_keywords()
    """
    sorted_items = sorted_items[:max_count]

    score_vals = []
    feature_vals = []

    for idx, score in sorted_items:
        score_vals.append(round(score, 3))
        feature_vals.append(self._feature_names[idx])

    results = {
        feature_vals[idx]: score_vals[idx]
        for idx in range(len(feature_vals))
    }
```

```
        return results
```

Декоратор timer

```
import functools
import datetime


def timer(func):
    """Print the runtime of the decorated function"""
    @functools.wraps(func)
    def wrapper_timer(*args, **kwargs):
        start = datetime.datetime.utcnow()
        value = func(*args, **kwargs)
        end = datetime.datetime.utcnow()
        print(f'{func.__name__}: {(end - start).total_seconds():.2f} seconds
elapsed.')
        return value
    return wrapper_timer
```

Класс Singleton

```
class Singleton(type):
    _instances = {}


    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton, cls).__call__(*args,
**kwargs)
        return cls._instances[cls]
```

Класс взаємодії з базою даних

```
import pymongo



class PymongoManager(object):

    def __init__(self, host='localhost', port=27017, usr=None, pwd=None,
db_auth=None, db='QADiploma'):

        mongo_connection_uri =
f'mongodb://{usr}:{pwd}@{host}:{port}/{db}?authSource={db_auth}' if (usr and
pwd and db and db_auth) else f'mongodb://{host}:{port}'

        self.client = pymongo.MongoClient(mongo_connection_uri)

        self.db = self.client[db] if db else None
```

Модуль Telegram

```
from telegram.ext import CommandHandler, Filters, MessageHandler, Updater


TOKEN = '820890263:AAGz37oB041P_F5DBl7kmYb4UfzYLmyFcdE'



class TelegramBot(object):

    def __init__(self, token=TOKEN):

        self.updater = Updater(token=token)


    @staticmethod

    def start(bot, update):

        update.message.reply_text('So, It begins... Ask me something.')


    @staticmethod

    def process_message(bot, update):

        update.message.reply_text(update.message.text)
```

```
    @staticmethod
    def error(bot, update, error):
        print(f'Update "{update}" caused error "{error}"')

    def prepare(self):
        dp = self.updater.dispatcher
        dp.add_handler(CommandHandler("start", self.start))
        dp.add_handler(MessageHandler(Filters.text, self.process_message))
        dp.add_error_handler(self.error)
```

Точка входу в систему

```
import functools
import datetime

from keywords_extractor import KeywordsExtractor


@timer
def main():
    ke = KeywordsExtractor()

    keywords = ke.extract_keywords("What is the size of an integer, bitch?")
    for key, rank in keywords.items():
        print(key, ': ', rank)


if __name__ == '__main__':
    main()
    print("I'm done")
```

Модуль вибору відповіді

```
import csv
import pathlib


from deeppavlov.agents.default_agent.default_agent import DefaultAgent
from deeppavlov.agents.processors.highest_confidence_selector import
HighestConfidenceSelector
from deeppavlov.contrib.skills.similarity_matching_skill import
SimilarityMatchingSkill
from deeppavlov.skills.pattern_matching_skill import PatternMatchingSkill


from pymongo_manager import PymongoManager
from timer import timer




class QADB(object):
    QA_DATA_PATH = './data/questions_answers.csv'
    FAQ_MODEL_PATH = './data/qa_model'


    def __init__(self, force_retrain=False):
        self.mongo_manager = PymongoManager()
        self.agent = None
        self._prepare(force_retrain=force_retrain)


    def _prepare(self, force_retrain):
        hello = self._create_hello_pms()
        bye = self._create_bye_pms()
        fallback = self._create_fallback_pms()
        faq = self._get_model(force_retrain=force_retrain)
        self.agent = DefaultAgent([hello, bye, faq, fallback],
skills_selector=HighestConfidenceSelector())
        qs = ['what is the size of the int variable in C', 'how to create an
array in C', 'how to check if character is digit in C', 'character string']
        questions = qs.copy()
```

```
        from keywords_extractor import KeywordsExtractor
        ke = KeywordsExtractor()
        for i in range(len(qs)):
            kw = ke.extract_keywords(qs[i])
            print(qs[i], ': ', kw)
            qs[i] = " ".join(kw.keys())
        print(qs)
        print(questions)

        print(faq(qs, [], []))
        print(faq(questions, [], []))
        # print(hello(qs, [], []))
        # print(bye(qs, [], []))
        # print(fallback(qs, [], []))
        # print(faq(qs, [], []))

    def _create_csv_from_mongodb(self):
        """
        Helper for _train_model().
        """
        answers = self.mongo_manager.db['answers'].find()

        questions_answers = []
        for a in answers:
            qa = {
                'Question':
self.mongo_manager.db['questions'].find_one({'_id': a['qid']})['body'],
                'Answer': f'{a["body"]}',
            }
            if 'additional_links' in a:
                qa['Answer'] = f'{qa["Answer"]}\n\nAdditional links:\n'
                for al in a['additional_links']:
                    qa['Answer'] = f'{qa["Answer"]}{al}\n'
            questions_answers.append(qa)
```

```
        with open(self.QA_DATA_PATH, 'w', newline='', encoding='utf-8') as
csvfile:
            fieldnames = ['Question', 'Answer']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for qa in questions_answers:
                writer.writerow(qa)


    @timer
    def _train_model(self):
        if not pathlib.Path(self.QA_DATA_PATH).exists():
            self._create_csv_from_mongodb()


        return SimilarityMatchingSkill(
            data_path=self.QA_DATA_PATH,
            x_col_name='Question',
            y_col_name='Answer',
            save_load_path=self.FAQ_MODEL_PATH,
            config_type='tfidf_autofaq',
            edit_dict={},
            train=True
        )


    @timer
    def _load_model(self):
        return SimilarityMatchingSkill(save_load_path=self.FAQ_MODEL_PATH,
train=False)


    def _get_model(self, force_retrain):
        model_path = pathlib.Path(self.FAQ_MODEL_PATH)
        if (not model_path.exists()) or (force_retrain is True):
            return self._train_model()
        else:
```

```
            return self._load_model()


    @staticmethod
    def _create_hello_pms():
        return PatternMatchingSkill(
            responses=[
                'Hello, my friend. Stay awhile and listen...',
                'Greetings, good master! Welcome to the Chat of the Rising
Sun.',
                'The warmth of life has entered my chat...',
                'Yo dawg',
                'Hello',
                'Hi',
                'Sup',
                'I greet you.',
            ],
            patterns=['\\b(?:hi|hello|hola|good day|good morning|good
evening|yo|sup)\\b', ],
            regex=True,
            default_confidence=0.75
        )


    @staticmethod
    def _create_bye_pms():
        return PatternMatchingSkill(
            responses=[
                'Goodbye',
                'Bye',
                'Cya',
                'See you later',
                'See ya',
            ],
            patterns=['\\b(?:bye|good bye|goodbye|cya)\\b', ],
            regex=True,
```

```
            default_confidence=0.75
        )

    @staticmethod
    def _create_fallback_pms():
        return PatternMatchingSkill(
            ['The university is immense... I cannot find an answer to this.'],
            default_confidence=0.4
        )


if __name__ == '__main__':
    db = QADB()
```

Додаток З

Копія презентації

ПРОГРАМНА СИСТЕМА ДЛЯ АНАЛІЗУ ЗАПИТІВ ПРИРОДНОЮ МОВОЮ

Виконав:
студент Романюк С.О.
ФПМ, КП-51
Науковий керівник:
доцент, к.т.н., Заболотня Т.М.

Постановка задачі

Мета:

Побудувати систему, яка могла б аналізувати запити, задані природною мовою, та надавати користувачу інформацію за цим запитом.

Відповідні задачі, які необхідно вирішити:

- підготувати базу даних запитів-питань і відповідей на них;
- проаналізувати характеристики питань;
- реалізувати алгоритм аналізу запиту природною мовою і пошуку відповіді;
- реалізувати графічний інтерфейс для взаємодії користувача із програмною системою;
- виконати тестування програмної системи.

Актуальність

- проблема самостійного пошуку і знаходження інформації;
- проблема витрачення часу людиною, яка дає відповідь на одні й ті самі питання.

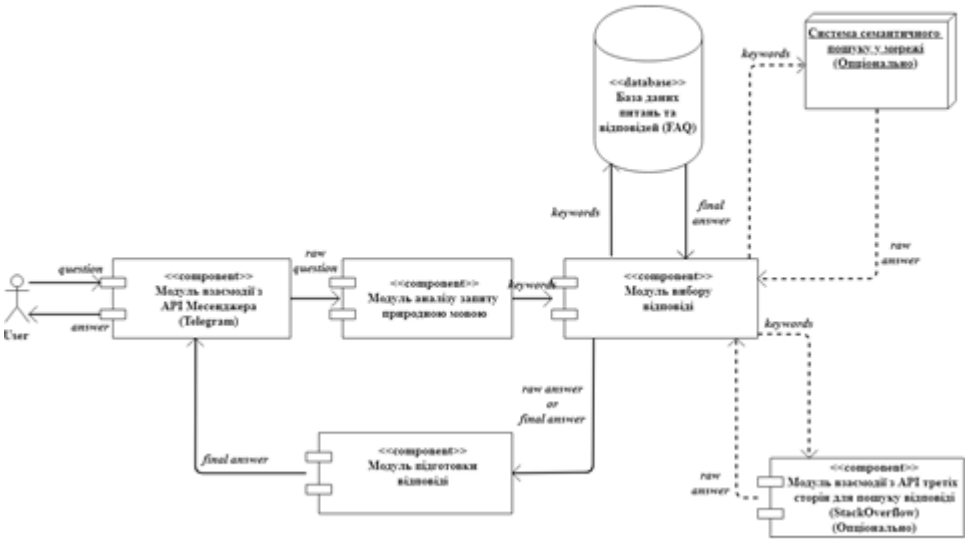
Об'єкт досліджень

Задачі, які вирішує NLP

Види QA систем. Мова запитів

Мова запитів: англійська.

Архітектура системи



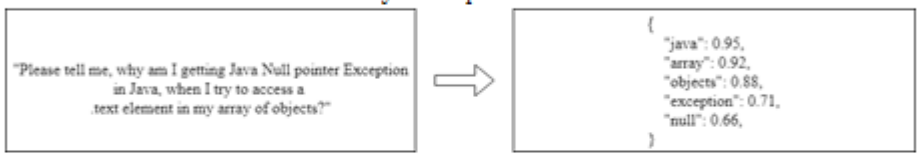
Попереднє оброблення питання

Етапи попереднього оброблення тексту питання:

- приведення до нижнього реєстру;
- видалення пунктуаційних та спеціальних символів;
- видалення цифр та чисел;
- лематизація – приведення слів до нормальної форми, виділення їх коренів.

Виділення ключових слів

Приклад виділення ключових слів з запиту. Алгоритм TFxIDF



Залежність рангу слова від його частоти серед усіх документів

Пошук відповіді. Джерела інформації. Значення впевненості.

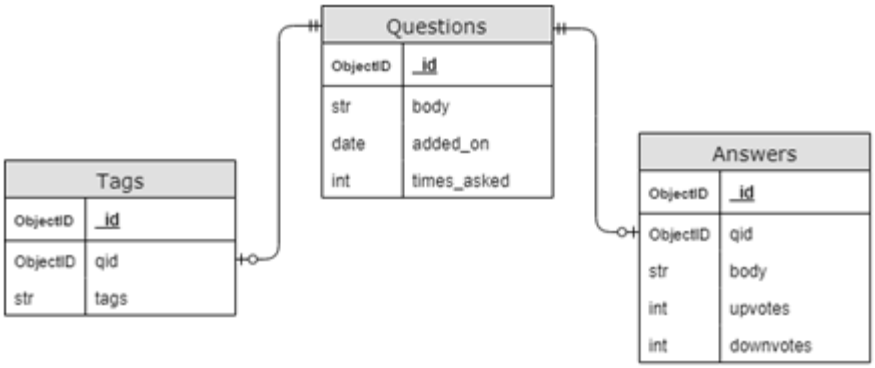
База даних часто задаваних питань та відповідей

Семантичний пошук у мережі

API сервісів третьої сторони

Архітектурні особливості модулю вибору відповіді

Схема бази даних питань та відповідей. Розширення БД.

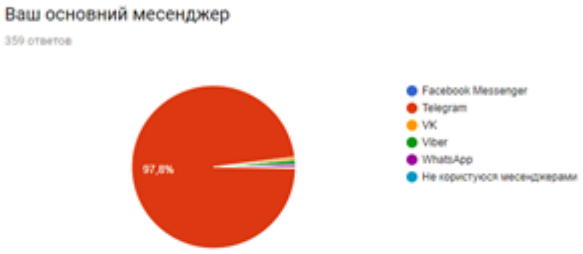


Зміна галузі системи

- використання відповідного тематиці набору даних для алгоритму TFxIDF;
- підбір нових питань та відповідей для БД;
- додатково: підбирання коректного для конкретної галузі сервісу у якості допоміжного модуля для пошуку відповіді.

Платформа для розміщення системи

Статистика месенджерів*

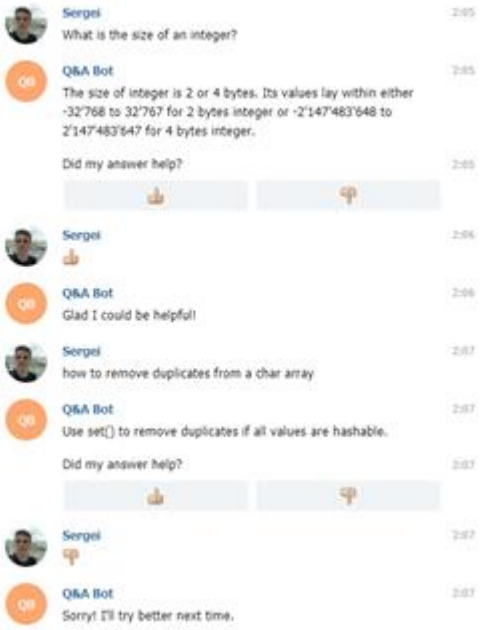


* повна статистика розміщена за посиланням (буде доступна після проходження опитування): https://forms.gle/1EVkxew1cayvnpQu9

Технології розробки

- Мова програмування: **Python**;
- СКБД: **MongoDB**;
- Платформа: **Telegram** *(API)*;
- Бібліотека *NLP*: **DeepPavlov**;
- Інші бібліотеки:
  - keras, nltk, scikit-learn, scipy;
  - Cython, numpy, pandas.

Приклади взаємодії



Результати

- ✓ Сформована база даних часто задаваних питань та відповідей на них;
- ✓ Проаналізовані характеристики питання;
- ✓ Реалізовано алгоритм підбору відповіді через наявні джерела даних;
- ✓ Проведена інтеграція з месенджером *Telegram* для забезпечення доступу до системи;
- ✓ Проведено тестування системи.

Дякую за увагу!

Додатково

- Щоб змінити платформу через яку надається доступ до системи (поставити інший месенджер), потрібно змінити (або додати новий) модуль взаємодії з API відповідного месенджера

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ПРОГРАМНА СИСТЕМА ДЛЯ АНАЛІЗУ ЗАПИТІВ ПРИРОДНОЮ МОВОЮ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ С.О. Романюк

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Програмна система для аналізу засобів природною мовою являє собою сервер, написаний на мові програмування *Python*. Доступ до функціоналу забезпечується через інтерфейс чат-боту на платформі *Telegram*.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1. Функціональна працездатність окремих компонентів системи;
2. Наявність доступу до бази даних часто задаваних питань та відповідей;
3. Взаємодія сервера з іншими опціональними модулями знаходження відповідей;
4. Забезпечення коректної обробки запитів від користувача;
5. Взаємодія сервера з API платформи *Telegram*;
6. Відповідність системи вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується комбінацією методів структурного тестування та методів поведінкового тестування. Такий підхід також називається тестуванням за принципом “сірої скриньки”. Перевіряється як сама кодова база, так і відповідний програмний продукт під час інтеракції на відповідність функціональним вимогам. Тестування відбувається на повній інтегрованій системі, тобто на рівні системного тестування.

Використовуються наступні методи:

1. Функціональне тестування, в тому числі тестування критичного шляху (тестування системи за стандартних умов користування нею);
2. Димове тестування;
3. Тестування взаємодії, в тому числі тестування сумісності та інтеграційне тестування;
4. Конфігураційне тестування (взаємодія з ботом через офіційні додатки *Telegram* на різних операційних системах).

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність створеної системи перевіряється шляхом:

1. Динамічного ручного тестування через введення некоректних символів та/або використання некоректної мови під час взаємодії з ботом;
2. Динамічного ручного тестування на відповідність функціональним вимогам;
3. Статичного тестування кодової бази;
4. Тестування доступності бота через офіційні додатки *Telegram* на різних операційних системах;
5. Тестування при конкурентному навантажені на систему з боку декількох користувачів;
6. Тестування стабільності роботи за різних умов.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ПРОГРАМНА СИСТЕМА ДЛЯ АНАЛІЗУ ЗАПИТІВ ПРИРОДНОЮ МОВОЮ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ С.О. Романюк

2019

ЗМІСТ

1. Загальні відомості про систему

Дана система представляє собою програмне забезпечення, яке можна використовувати для аналізу запитів природною мовою.

Інтерактування з системою відбувається через використання боту, розміщеного на платформі *Telegram* [1].

2. Огляд платформи, на якій розміщений чат-бот

Telegram – це популярний багато платформний месенджер, який дозволяє обмінюватися повідомленнями і медіафайлами.

Для доступу до свого облікового запису у межах системи та функцій самого месенджера можна використовувати майже будь-яку операційну систему. Окрім того, що є доступними офіційно розроблені самою командою месенджера *Telegram* застосунки для операційних систем *Android, iOS, Linux, MacOS, Windows* та *Windows Phone*, доступитися до системи можна через Web-версію за допомогою браузера. Також, для популярного браузеру *Google Chrome* є окремо створений застосунок.

Умовою доступу до месенджеру за допомогою будь-яких офіційних та неофіційних застосунків є доступ до мережі Інтернет. Також, так як користувач ідентифікується у межах месенджеру за допомогою свого мобільного телефону, а підтвердження реєстрації відбувається за допомогою SMS, то, на момент реєстрації, слід мати телефон поруч.

Таким чином, щоб отримати доступу до створеної у межах даного дипломного проекту системи, користувач має бути зареєстрованим та автентифікованим у месенджері і мати базові навички користування ним.

3. Процедура авторизації на платформі *Telegram*

У даному пункті передбачається, що користувач вже є зареєстрованим на платформі. Якщо ж ні, то за посиланням [2] наведені процедури реєстрації через найбільш популярні операційні системи.

Для того, щоб авторизуватися на платформі *Telegram*, перш за все слід обрати систему, яка є для користувача найбільш зручною, та завантажити додаток. Приклад у межах даного керівництва користувача буде приведений для браузерної версії платформи, яка є загальною для усіх платформ.

Отже, спочатку, треба перейти за посиланням [3]. Після цього, слід ввести свій мобільний номер телефону та натиснути кнопку *"Next"*, а у модальному вікні, що з'явилося, перевіривши коректність введеного номеру, натиснути кнопку *"OK"* (рис. 1).

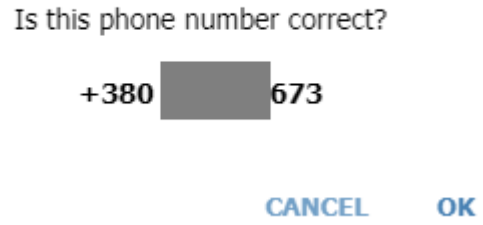


Рис. 1. Підтвердження коректності введеного номеру

Після цього, так як користувач вже є зареєстрованим у додатку, йому надійде повідомлення з кодом підтвердження автентифікації у веб-версії додатку. Повідомлення надійде у вигляді звичайного повідомлення або на обліковий запис користувача, якщо він встановлював мобільний або десктопний додаток, або у вигляді SMS (рис. 2).

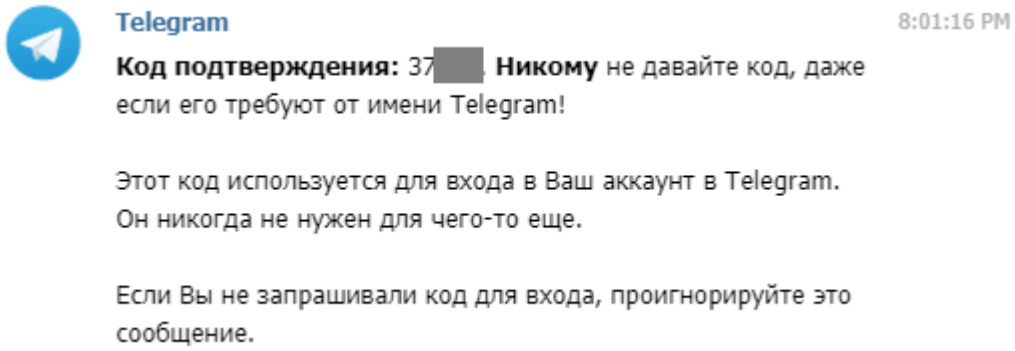


Рис. 2. Код підтвердження для входу через Web-версію

Отриманий код слід ввести у поле, яке буде перед користувачем (рис. 3).

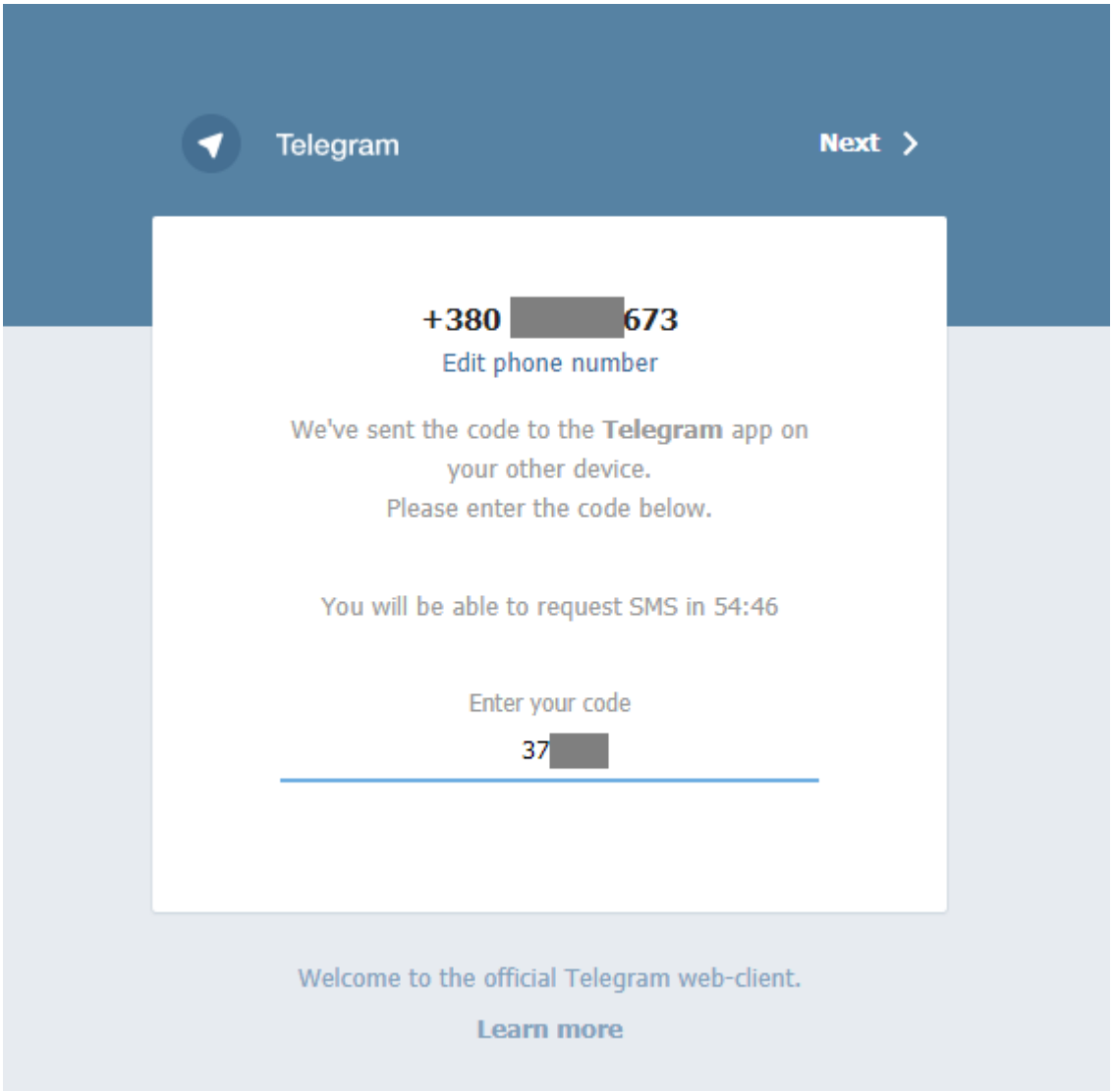


Рис. 3. Введення отриманого коду підтвердження

Після введення правильного коду підтвердження, користувача буде направлено на сторінку з повідомленнями. На цьому процес авторизації завершено.

4. Знаходження бота у межах платформи *Telegram*

Інтеракція з ботом відбувається таким саме чином, як і користування звичайним приватним чатом у межах месенджеру. Інтерфейс чату у межах Web-версії виглядає наступним чином (рис. 4).

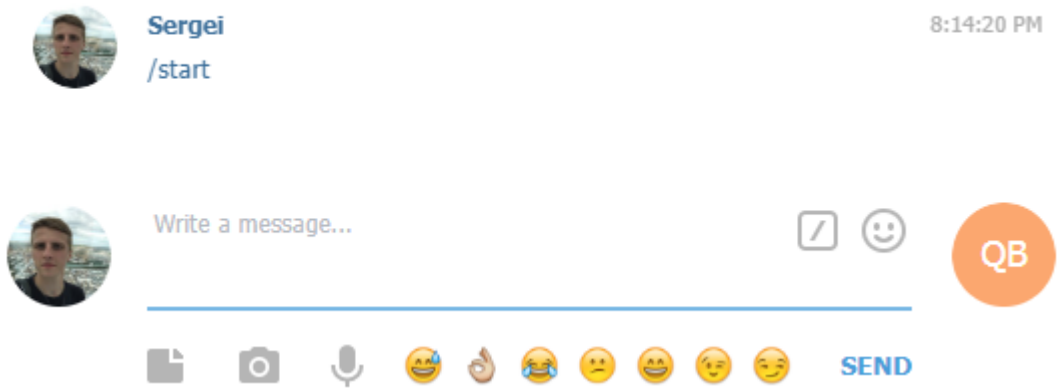


Рис. 4. Інтерфейс чату з ботом

Отже, для початку “спілкування” з ботом, слід ідентифікувати його у межах платформи. Для цього слід у полі *“Search…”* ввести унікальний ідентифікатор бота: *@qa_prog_bot*. Він відобразиться у результатах пошуку під іменем “Q&A Bot” (рис. 5).

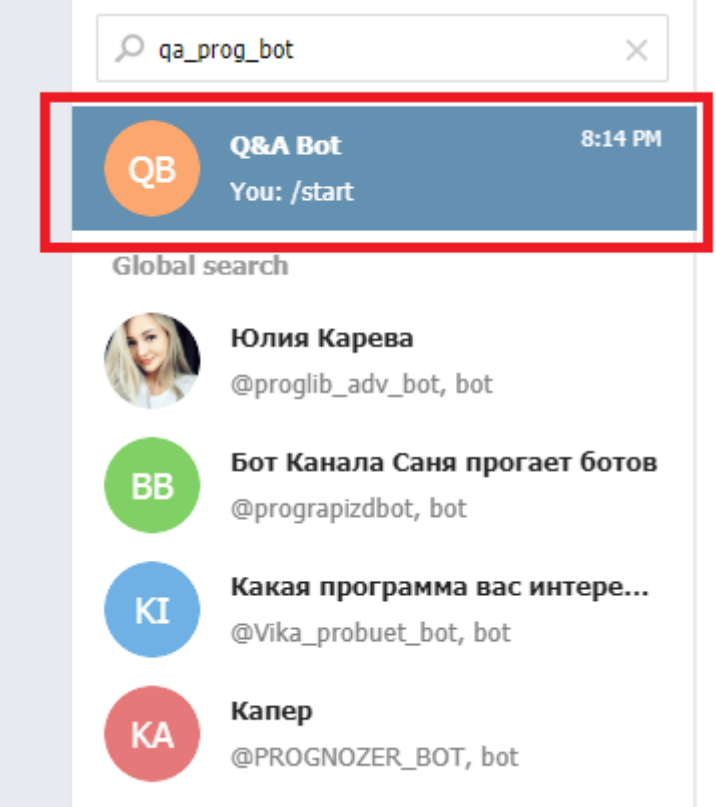


Рис. 5. Пошук бота за його унікальним ідентифікатором

Після знаходження бота, слід натиснути на нього. Користувач буде переведений у діалогове вікно з даним ботом.

5. Процедура відправлення запиту боту

Щоб відправити боту запит, слід перейти до діалогу з ним, як описано у розділі 4.

Тепер, перш за все, щоб мати змогу саме вести діалог з ботом, а не монолог, треба відправити боту початкове повідомлення. Задля того, щоб захистити користувачів системи від спам-ботів, *Telegram* не дозволяє ботам бути ініціаторами діалогу з реальними користувачами.

Зазвичай, текст першого повідомлення боту – це команда *"/start"*. Саме таке повідомлення буде відправлено боту, якщо користувач не вів переписку з ботом раніше і натиснув кнопку *"START"*, як пропонується інтерфейсом (рис. 6).

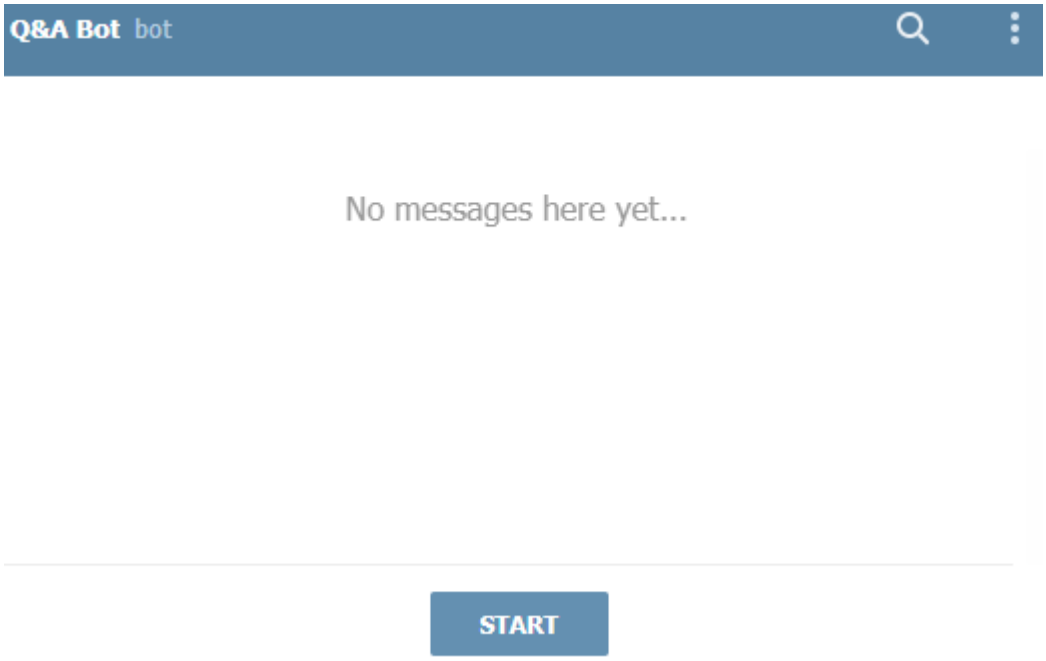


Рис. 6. Початок діалогу з ботом. Кнопка "START"

Отже, після натиснення даної кнопки, буде відправлене початкове повідомлення і боту можна буде задавати питання (вести з ним переписку у вигляді діалогу) (рис. 7).

Рис. 7. Початок діалогу з ботом. Відправлена команда "/start"

Процес поставлення питання дуже простий. Слід просто запитати бота про щось і дочекатися відповіді (рис. 8).

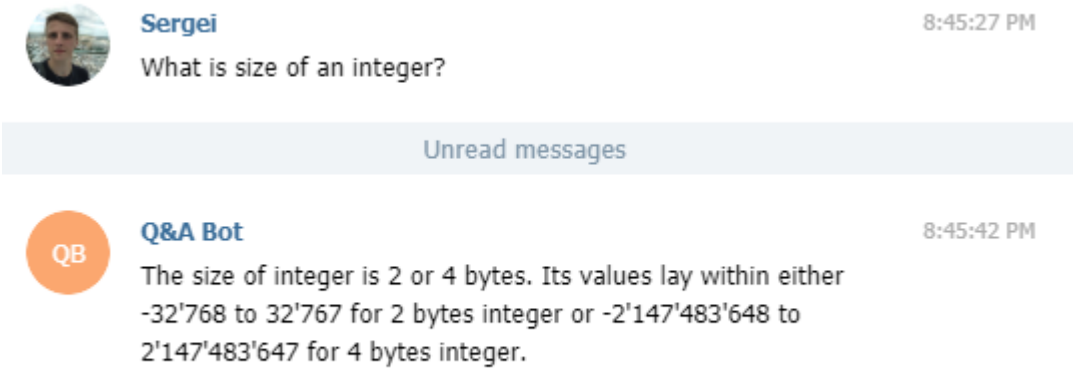


Рис. 8. Відповідь бота на поставлене питання

Окрім цього, у бота наявні декілька наперед підготовлених відповідей для ряду повідомлень. Наприклад, повідомлень привітання (рис. 9).

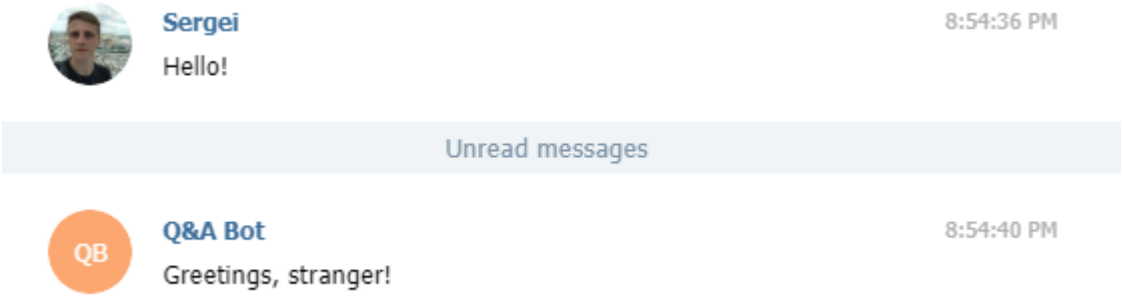


Рис. 9. Відповідь бота на привітання

Зарезервованими є також реакція на подяку та прощання.

Якщо ж боту не вдалося знайти відповідь на поставлене запитання, він також нотифікує про це відповідним повідомленням:

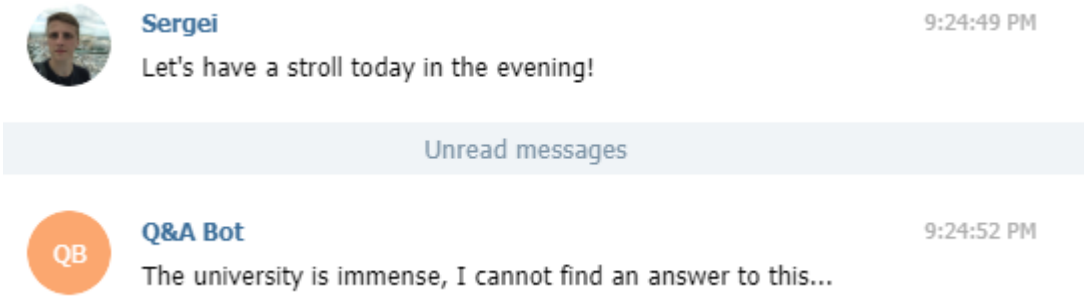


Рис. 10. Повідомлення бота про те, що відповідь на задане питання не була знайдена

6. Особливості формування запитів для бота

На момент підготовки даного керівництва двома основними джерелами для пошуку відповідей на поставлений запит (питання) є невелика підготовлена база даних відповідей на часто задавані студентами питання та додатковий модуль семантичного пошуку у мережі, створений у рамках іншої дипломної роботи.

Пріорітетним джерелом відповідей є сама підготовлена база. Якщо в ній не було знайдено відповіді на поставлене питання, буде активізований додатковий модуль семантичного пошуку, а також інші опціональні модулі за наявності. Підтримка інших джерел може бути додана у майбутньому.

Також, єдина мова, доступна під час інтеракції з ботом на момент написання даного керівництва – англійська. Не слід задавати питання іншими мовами, – бот не зможе знайти на ці питання відповідь. Відповідно, мова, якою локалізовані відповіді, – також англійська.

Питання слід формувати у одному повідомленні. Кожне повідомлення трактується ботом як окремий запит або питання.

Слід очікувати, що через те, що база даних бота може модифікуватися (туди можуть додаватися нові відповіді або видалятися

нерелевантні), то на одне й те ж питання, задане боту у різні проміжки часу, може бути отримана різна відповідь, – це є нормальним.

ПРИМІТКИ

1. Telegram Messenger [Електронний ресурс]. — Режим доступу: https://telegram.org/
2. Регистрация в Телеграмм на русском [Електронний ресурс]. — Режим доступу: http://telegram.org.ru/494-registraciya-v-telegramm.html
3. Telegram Web [Електронний ресурс]. — Режим доступу: https://web.telegram.org